



محاضرات

C++

للأستاذ/

بســــــــام الهــــــاملي

إعداد الطالب:

هشام يحيى دلال

الإهداء

اللهم إني أردت بهذا العمل وجهك الكريم فتقبل مني، اللهم من استفاد من هذا الكتاب بمعلومة فاكتب لي بها حسنة وإن تضاعف لي فإنك أنت الحليم الكريم، ومن صححت لديه فكرة كانت خاطئة فأمحو عني سيئة وإن تضاعف فإنك أنت العفو الغفور.

اللهم اغفر لي ولمن علمني هذا العلم وأطل في عمره..

اللهم زدني علماً وانفعني به، اللهم إني أعوذ بك من علم لا ينفع وقلب لا يخشع وصلاة لا ترفع وصدقة لا تقبل وأعوذ بك من الرياء والنفاق، والكبر والعجب، والبخل والشح، والعجز والكسل، وأرذل العمر، وعذاب القبر، وفتنة المحيا والممات، وفتنة المسيخ الدجال.

اللهم إني اهدي هذا الكتاب لجميع طلاب العلم من المسلمين فاحفظه من التلف أو الضياع أو الإهمال..

اللهم آمين،،،

في حالة وجود أخطاء:

أرجو شاكراً أن تساهم معي في تصحيح الأخطاء الإملائية والأخطاء في الفكرة أو الكود وغيرها عبر إرسالها على البريد: hishamdalal@gmail.com، مع ذكر رقم الفصل الذي يحتوي على الخطأ.

الفصل
1

مقدمة عن البرمجة
Introduction

البرمجة (Programming):
عبارة عن مجموعة من الأسس والمبادئ والنظريات التي تتلائم مع البيئات العملية "مبادئ علم الحاسوب".

البرنامج (Program):
- تعريف عام : مجموعة من التعليمات "Codes" والأوامر المرتبة لحل مشكلة معينة.
- أو : مجموعة من الأكواد المكتوبة بلغة من لغات البرمجة.

الحزم (Package):
مجموعة من البرامج الجاهزة المتكاملة والمترابطة فيما بينها، والتي تؤدي وظائف متعددة وتعمل تحت بيئة واحدة.
برامج جاهزة: يعني لا يتم التعديل عليها فهي صيغة نهائية مثل : windows, office package

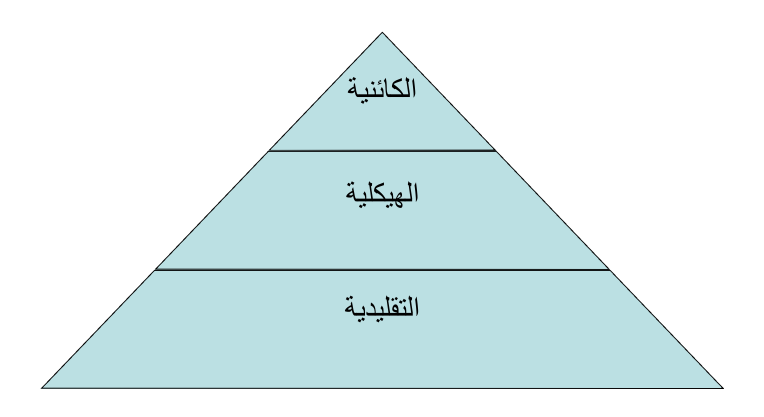
أنواع البرامج Program types:
١- أنظمة التشغيل (OS).
٢- لغات البرمجة (Programming Languages).
٣- التطبيقات (Applications).
٤- المفسرات/المترجمات (Compilers).

مستويات البرمجة:
وهي مستويات يمر عليها المبرمج خلال رحلته التعليمية البرمجية:

١) التقليدية: وهي مرحلة كتابة برامجه بطريقة عشوائية لا تعتمد على أسس ونظريات وهيكلة، فما يهم المبرمج هو الوصول لحل المشكلة فقط.
عيوبها:
- لا يمكن اكتشاف الخطأ في البرنامج.
- لا يمكن تطوير البرنامج بسهولة.
- تطوير البرنامج يزيد حجم البرنامج بشكل كبير.

٢) الهيكلة: وهي مرحلة يقوم فيها المبرمج بتقسيم برنامجه إلى هياكل تساعد على اكتشاف الأخطاء والتطوير كما أعطت إمكانيات كثيرة.

٣) الكائنية: وهي مرحلة يقوم فيها المبرمج بهيكلة برنامجه إلى كائنات كل كائن يتخصص في حل مشكلات في مجال معين، ويحتوي الكائن على طرق "دوال" مترابطة تجعل من الكائن يمتلك ذكاء في تحديد الاستجابة المطلوبة تلقائياً.



مستويات لغات البرمجة (Programming Language Levels):

١. المستوى الأدنى (Lowest Language Level (L.L.L:

يتعامل هذا المستوى مع الدوائر المنطقية ويتكون من :

- لغة الآلة (Assembler).
- لغة التجميع (Micro Assembly).

عيوبه:

- صعوبة كتابة البرمجيات أو فهمها لأنها تتعامل مع رموز بالنظام الثنائي "0/1".
- لغة قريبة من الآلة وبعيدة عن الإنسان.
- تحتاج إلى متخصصين في الحاسوب.

ملاحظة:

نظام الإدخال (ثنائي)
نظام الحفظ (عشري).
نظام العرض (سادس عشر).

٢. المستوى المتوسط (Midst Language Level (M.L.L:

ظهر هذا المستوى ثم اختفى بسرعة بسبب سرعة التطوير إلى المستويات العليا حيث دمجت وصنفت لغاته في المستوى العالي ويتكون من اللغات التالية:

- C.
- C++.

مميزاته:

- قريبة من الحاسوب ومن الإنسان.

٣. المستوى العالي (Highest Language Level (H.L.L:

تتكون من :

- Basic.
- Pascal (تستخدم في المجال العلمي).
- Fortran (تستخدم في المجال الفيزيائي والرياضي).
- Cobol (تستخدم في المجال التجاري).
- Java.

مميزاته:

- قريبة جداً من لغة الإنسان.
- سهلة الفهم والكتابة.
- لغات هيكلية.
- لغات متخصصة (كل لغة تهتم بجانب معين، وبالتالي يمكن الاستفادة من اللغات المختلفة بحسب نوع المشكلة المراد حلها).

الفصل
2

مقدمة عن C++
Introduction C++

نبذة تاريخية:
أول لغة ظهرت هي الأسمبلي للتعامل مع البوابات المنطقية ومبادئ الحوسبة، ثم ظهرت مبادئ لغة C في الأربعينيات.

مميزات لغة C :

- لغة كاملة وشاملة (تعتبر أم لغات البرمجة).
- بيئة تطويرية Integrated Development Environment (IDE) تحوي كل الأدوات.
- إنشاء برامج مساعدة لا تعتمد على واجهات (Interface) تتميز بأنها سرية وآمنة وقوية.
- مكتبات تنفيذ المشروع.
- MFC صفوف ميكروسوفت التأسيسية Microsoft foundation class تساعد في إنشاء الواجهات Graphic user interface (GUI) (واجهات التخاطب مع المستخدم).
- أدوات البناء Built tools تساعد في استخدام الأدوات الموجودة في C++.

الشكل العام للبرنامج:

Header Files	1.	#include <library_name.h>	١. استيراد المكتبات
	2.	Public Declaration	٢. منطقة التصاريح العامة
Program Body	3.	Main ()	٣. الدالة الرئيسية
	4.	{	٤. بداية الدالة الرئيسية
	5.	Private Declaration	٥. منطقة التصاريح الخاصة
	6.	Statements.. Statements.. Statements..	٦. جمل برمجية
	7.	}	٧. نهاية الدالة الرئيسية

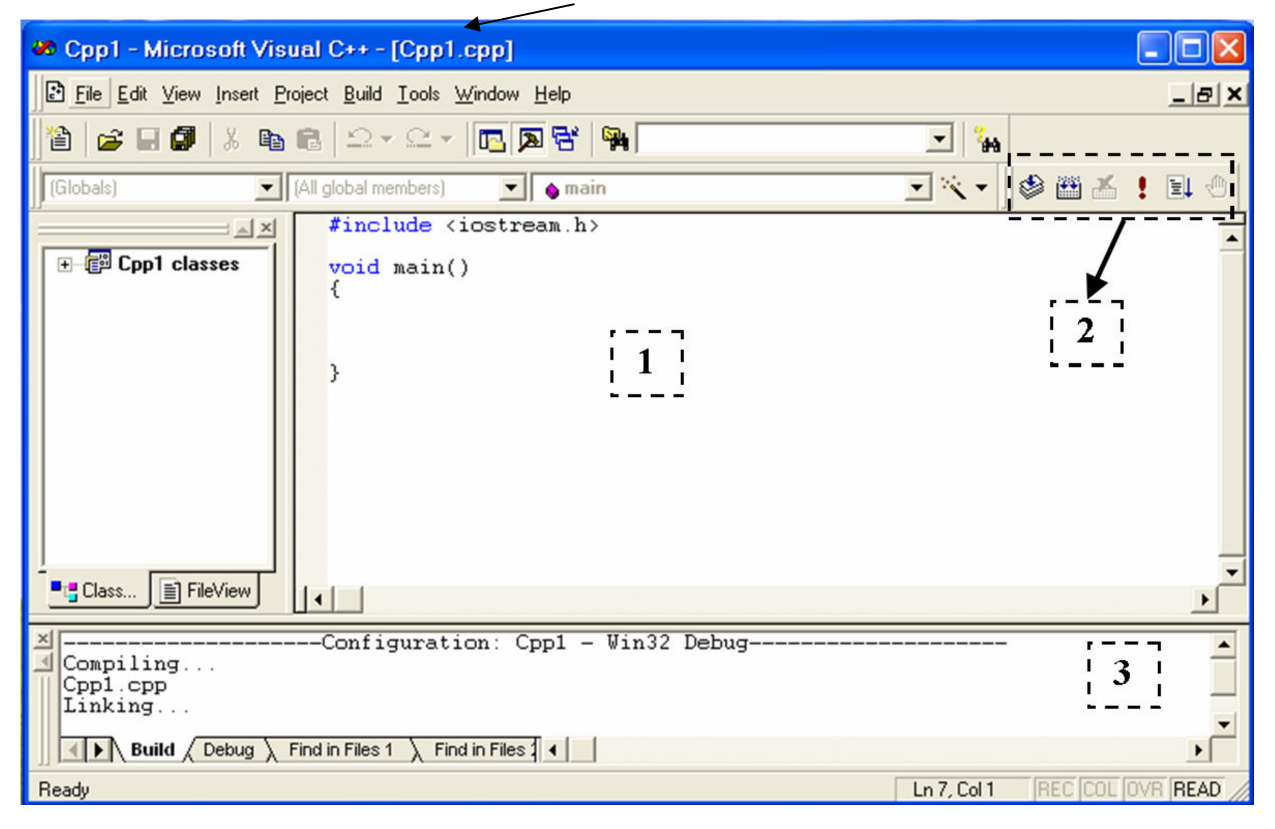
الفصل

3

واجهة بيئة C++
C++ Interface

واجهة البرنامج:

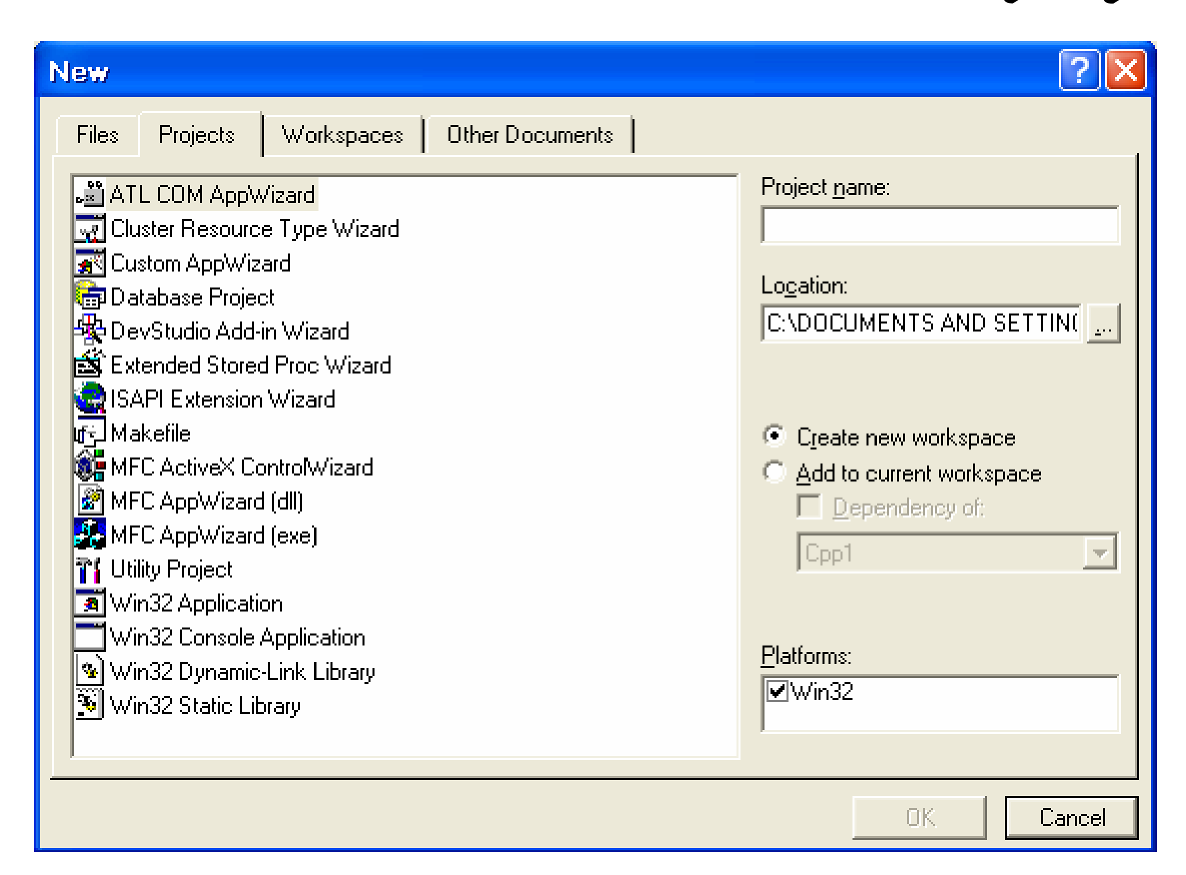
اسم الملف المصدري مع الامتداد cpp.



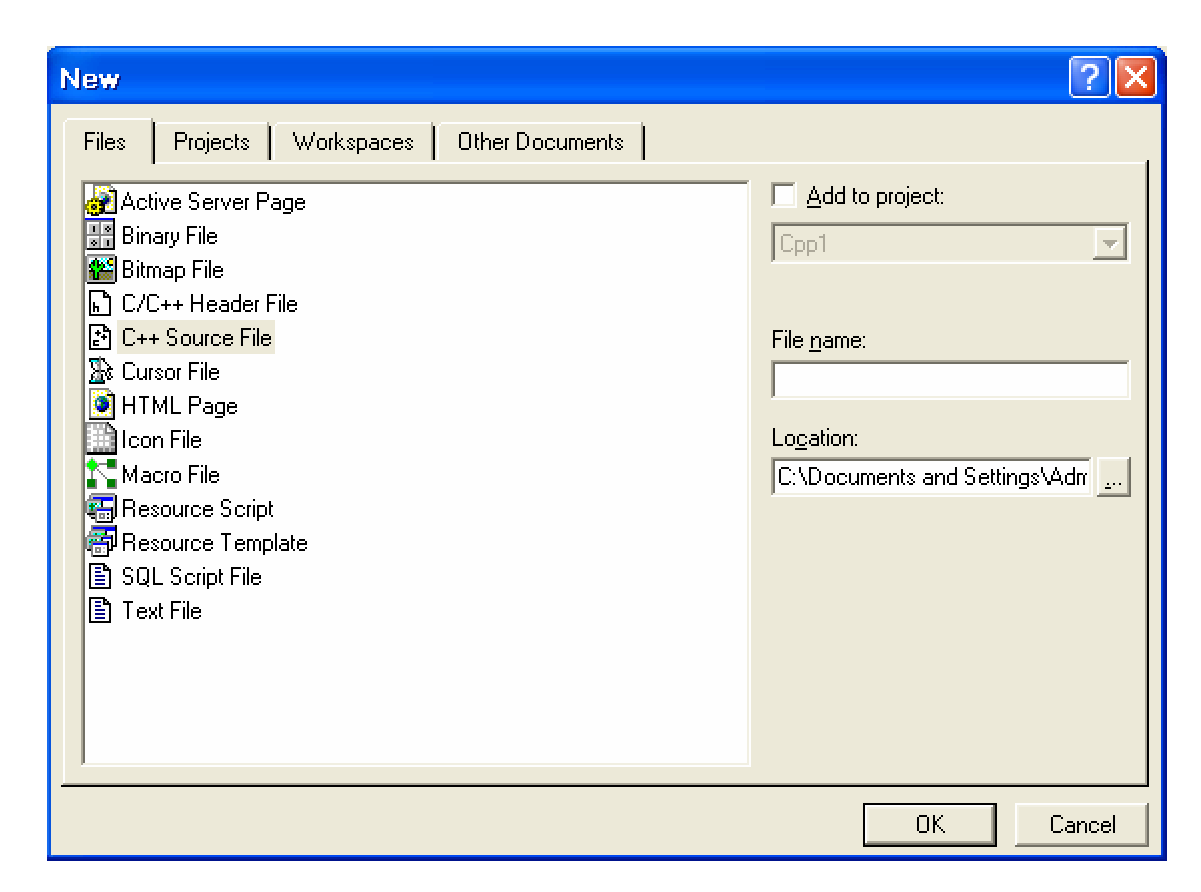
أهم الأجزاء:

[١] مكان كتابة الكود.
[٢] تنفيذ وعرض البرنامج، ويتكون من مرحلتين: مرحلة تكوين ملف مصدري، ثم ترجمة البرنامج للتنفيذ.
[٣] منطقة عرض الأخطاء، ومن خلالها يتم تحديد مكان الخطأ Error مع نوعية الخطأ.

أنواع المشاريع:



PROJECTS	المشروعات
Database Project	قواعد بيانات
MFC ActiveX ControlWizard (ocx)	ملفات مساعدة (لها خصائص وليس لها واجهات)
MFC AppWizard (dll)	ملفات مساعدة (لها خصائص مثل النموذج) تساعد في تصميم الواجهات.
MFC AppWizard (exe)	ملفات مساعدة (تحوي النوعين dll, ocx)
Utility Project	برامج خدمية
Win32 Application	تصميم تكوين مشروع من عدة تطبيقات
Win32 Console Application	تطبيق شاشة سوداء



FILES	الملفات
Active Server Page	صفحات انترنت تفاعلية ASP
Binary File	ملفات ثنائية (0/1)
Bitmap File	خريطة بيتات تنتج صورة
C/C++ Header File	مكتبات "إنشاء مكتبات" بامتداد h.
C++ Source File	برامج ++C "التي سنستخدمها" ملفات مصدرية
Cursor File	إنشاء صور مؤشر الفأرة
HTML Page	إنشاء صفحات ويب في بيئة html
Icon File	إنشاء أيقونة (رمز)

الفصل
4

مكونات C++
C++ Components

المكتبات:

١) iostream.h (مكتبة عامة لأوامر الإدخال والإخراج)
٢) stdio.h (مكتبة عامة "أقدم مكتبة" لأوامر الإدخال والإخراج)
٣) conio.h (مكتبة دوال أوامر الشاشة)
٤) math.h (مكتبة الدوال الرياضية)
٥) String (مكتبة دوال معالجة النصوص)

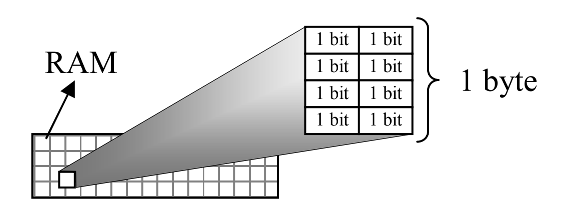
ملاحظة:

في أسماء المكتبات مثل (iostream.h , stdio.h) :
- i : يعني أوامر الإدخال ودوال الإخراج (Input).
- o : يعني أوامر الإخراج ودوال الإدخال (Output).
- s : البعض يقصد بها Standard والبعض يصفها بـ System.

أنواع البيانات:

INTEGER	Bytes	REAL	Bytes	STRING	Bytes	LOGIC	Bytes
Short	2	Float	4	Char	1	Bool	1
Int	4	Double	8	String	8		
Long	4						

هذه المسميات موجودة في الذاكرة العشوائية (RAM)، لكل نوع تقسيم "حجم" معين.



النوع	الحجم
Bit	2 (0/1)
Byte	8 bit
Kilobyte	1,000 byte
Megabyte	1,000,000 byte
Gigabyte	1,000,000,000 byte

المتغيرات:

أسماء كلمات ليست من كلمات اللغة المحجوزة، يصطلحها المبرمج (مستخدم بيئة تطوير لغة البرمجة) بغرض تخزين قيم في الذاكرة الرام لتمكين المبرمج من إجراء العمليات المختلفة على المتغيرات. فالمتغير يحفظ في موقع في الذاكرة، وإذا أراد المبرمج أن يزيد أو ينقص قيمة المتغير فيستطيع من خلال اسم المتغير.

الشكل العام:

```
DataType VariableName;
```

أمثلة

```
1. int x;
2. char ch;
```

إسناد قيم للمتغيرات:

الكود	الشرح
1. int x = 5 ;	تعريف متغير x يحمل قيمة ابتدائية 5.
2. x = 10 ;	إسناد قيمة جديدة لـ x.
3. x = 20 ;	تغيير القيمة السابقة بقيمة جديدة أخرى.
4. x = 3 + 5 ;	إسناد ناتج القيمة الحسابية[1] للمتغير (المتغير سيحمل القيمة ٨)
5. cin >> x ;	تغيير القيمة أثناء التشغيل من قبل المستخدم[2].
6. char ch = ' y ' ;	إسناد قيمة حرفية لمتغير (يكتب داخل تعليق مفرد ' ')

شروط تعريف المتغيرات:

١. لا يبدأ برقم أو عملية حسابية أو رمز ما عدا _ (underscore).
٢. ألا يحتوي على عملية حسابية أو رمز أو فراغ.
٣. ألا يزيد عن 255 حرفاً.

(1) العمليات الرياضية في الفصل السابع.
(2) استقبال القيم من المستخدم أثناء التشغيل في الفصل التالي.

القيم الابتدائية الثابتة والمتغيرة (Initialization & Const)

القيمة الابتدائية :

هي قيم تسند للمتغير بمجرد تعريفه وهي نوعين:

القيم المتغيرة :

يمكن تغييرها في البرنامج من قبل المبرمج أو أثناء التشغيل "run mode" باستخدام "cin".

القيم الثابتة (const):

لا يمكن تغييرها بأي حال من الأحوال.

فائدتها : حماية القيم التي نحتاجها كما هي ولا نريد أن يتم تغيير قيمتها بالخطأ.
مثل : قيمة الثابت π (3.14)

مثال لمتغير:

```
1. int x = 5 ;
2. x = 10 ;
3. x = 20 ;
4. cin >> x ;
```

تعريف متغير x يحمل قيمة ابتدائية 5.
إسناد قيمة جديدة لـ x.
تغيير القيمة السابقة بقيمة جديدة أخرى.
تغيير القيمة أثناء التشغيل من قبل المستخدم.

مثال لثابت:

```
1. const int x = 5;
2. x = 10 ;    // Error
```

تعريف ثابت يحمل قيمة ثابتة 5.
إسناد قيمة جديدة يولد خطأ في تنفيذ البرنامج.

التعليقات Comments:

عبارة عن توضيحات يكتبها المبرمج لا تدخل في تركيب البرنامج (لا ينفذها المترجم).

وتكون على شكلين :

- تعليق السطر الواحد :

```
1. // This is a comment
2. // And this is another comment
```

- تعليق الأسطر المتعددة:

```
1. /*
2. This is a comment
3. In tow lines
4. */
```

```
1. /* This is
2. a comment
3. in three lines */
```

الفصل
5

أنواع المكتبات
Libraries type

المكتبة IOSTREAM

تم دمج مكتبتين ضمن هذه المكتبة :
- Istream مكتبة الإدخال.
- Ostream مكتبة الإخراج.

أوامر الإدخال والإخراج:

تحتوي مكتبة iostream.h على دوال منها cin و cout ويتم استخدامهما كالتالي:

إدخال قيمة للمتغير x من قبل المستخدم:	1. cin >> x ;
إخراج قيمة المتغير إلى الشاشة:	2. cout << x ;
أرقام ورموز وحروف (انجليزية) باستخدام شرطة مزدوجة " "	3. cout << " نص " ;
حرف واحد باستخدام شرطة مفردة ' '	4. cout << ' c ' ;

مثال ١:

```
1. #include <iostream.h>
2. Main()
3. {
4.     int x ;
5.     cin >> x ;
6.     cout << " X value is: " << x ;
7. }
```

مثال ٢:

```
1. #include <iostream.h>
2. Main()
3. {
4.     int x , y ;
5.     cin >> x >> y ;
6.     cout << " first value is: " << x  << " second value is: " << y ;
7. }
```

المحارف الخاصة:

هي رموز محجوزة تعبر عن الحروف غير المطبوعة وتستخدم مع الدوال مثل (cout) و (printf) وتكون ضمن إشارتي تنصيص مزدوجة أو مفردة.

المحرف	المعنى	توضيح
\n	New line	سطر جديد
\t	8 Spaces (Tap)	٨ مسافات فارغة
\b	Backspace	الرجوع للخلف
\a	Sound "beep"	إصدار صوت من الجهاز

مثال:

```
1   cout << '\n';

2   cout << "Ahmed \t 20";

3   cout << "khaled\nSaleh";
```

الناتج: النزول إلى سطر جديد فارغ

الناتج :

```
Ahmed        20
```

الناتج:

```
khaled
Saleh
```

دوال تقوم بعمل المحارف الخاصة:

تستخدم مع الدالة (cout) .

الدالة	المعنى	توضيح
endl	New line	سطر جديد
ends	8 Spaces (Tap)	٨ مسافات فارغة

مثال:

```
1   cout << "Ahmed" << ends << "20";

2   cout << "khaled" << endl << "Saleh";
```

الناتج :

```
Ahmed        20
```

الناتج:

```
khaled
Saleh
```

المكتبة Stdio.h :

تحتوي على دالتين مهمتين :

- printf وهي دالة خاصة بعمليات الإخراج.
- scanf وهي دالة خاصة بعمليات الإدخال.

١) PrintF :

تتميز printf عن cout أنه يمكن كتابة النص والمتغير في نفس السطر بدون الحاجة لمعامل الاخراج (<<) ولكن بدلاً عن كتابة اسم المتغير يكتب التمثيل الديناميكي للمتغير حسب نوعه مسبوقاً بالرمز %:

النوع		التمثيل الديناميكي	ملاحظة
int	←	%d	أول حرف من من digital
char	←	%c	أول حرف من char
string	←	%s	أول حرف من string
float	←	%f	أول حرف من float

الصيغة العامة:

طباعة نص:
```
Printf(" النص ");
```
طباعة نص مع متغير:
```
printf("التمثيل الديناميكي النص ", var-name);
```
طباعة أكثر من متغير:
```
printf("تمثيل ديناميكي تمثيل ديناميكي", var-name1, var-name2);
```

أمثلة:

	الناتج:
1. Printf(" welcome ");	
2. int a = 255 ;	
3. printf(" Area = %d " , a);	Area = 255
4. int x = 1; int z = 2;	
5. printf(" v1 = %d and v2 = %d ", x , z);	v1 = 1 and v2 = 2

٢) ScanF :

يمكن من خلال هذه الدالة استقبال المتغيرات من المستخدم وإسنادها للمتغيرات تماماً مثل (cin).

الصيغة العامة:

استقبال قيمة من المستخدم:

```
scanf (" التمثيل الديناميكي ", &var-name);
```

استقبال قيمتين من المستخدم:

```
scanf ("تمثيل ديناميكي تمثيل ديناميكي", &var-name1, &var-name2);
```

ملاحظة:
يجب كتابة الرمز & قبل أي متغير.

أمثلة:

```
1. int x ; char y;
2. scanf ("%d "  ,  &x );
3. scanf ("%d%c ",  &x  , &y );
```

واجب :
ابحث عن الدوال : getch() و puts().

المكتبة Math.h :

تحتوي على دوال رياضية كثيرة مثل:

الدالة	الرمز الرياضي	توضيح
abs(x)	$\| x \|$	الأعداد الحقيقية
sin (x)		جا س
cos (x)		جتا س
tan (x)		ضا س
sinh (x)		جا-١ س
cosh (x)		جتا-١ س
tanh (x)		ضا-١ س
pow (x , y)	x^y	الاس
exp (x)	e^x	e
sqrt (x)	$\sqrt{x}$	الجذر
log (x)	Log x	اللوغاريتم
ceil (x)		تقريب الكسور للأعلى
floor (x)		حذف الكسور

مثال:

```
cout << ceil( 3.44 );
cout << ceil( -3.77);
cout << floor( 3.44 );
```

الناتج:

4
-3
3

المكتبة String :

توفر نوع من أنواع البيانات وهو (string) الذي يقبل تخزين مجموعة حروف ورموز وأرقام كنص في متغير واحد.

تختلف المكتبة عن سابقاتها، فمن أجل تعريف متغير x من نوع string يجب :

١. تضمين المكتبة string.
٢. إلغاء h. من اسم المكتبة "باستثناء المكتبات القديمة ".
٣. تحديث المكتبات..

مثال:

```
1.  #include <string>
2.  #include <iostream>
3.  #include <stdio.h>
4.
5.  using namespace std;

6.
7.  main ()
8.  {
9.      string s ;
10.     s = " Bassam ";
11.     cout << s;
12. }
```

تضمين المكتبة string بدون كتابة اللاحقة h.
تضمين مكتبة جديدة لذا لا نكتب اللاحقة h.
تضمين مكتبة قديمة لذا تبقى اللاحقة مثل stdio.h, math.h

تحديث المكتبات حيث std يحوي أوامر جديدة للمكتبات القابلة للتحديث "المكتبات الجديدة"

تعريف متغير نصي
إسناد قيمة نصية للمتغير
طباعة المتغير (عرض المتغير على الشاشة).

الفصل

6

تنسيق مخرجات البرنامج
Format Outputs

تنسيق الشاشة:

يمكن تحديث المكتبات بدون تضمين مكتبة string ، ومن مميزات تحديث المكتبات الحصول على دوال system التي تمكننا من تنسيق شاشة الإخراج وكذلك استخدام جميع أوامر نظام (DOS).

مثال:

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  main ()
5.  {
6.      system("color f0") ;
7.      cout << "new colors";
8.      system("pause");
9.  }
```

تضمين مكتبة مع حذف اللاحقة h.
تحديث المكتبات

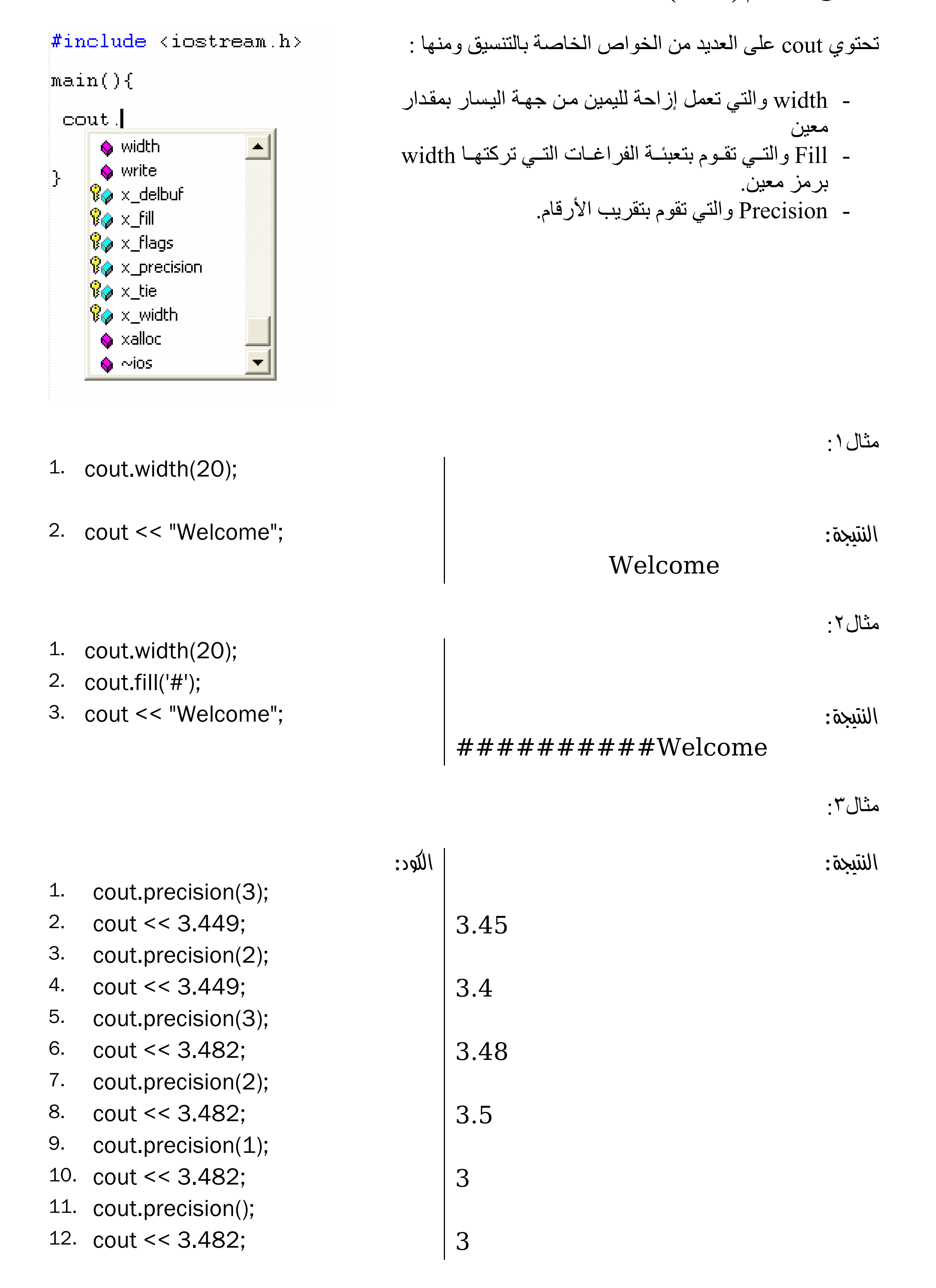
تنسيق لون النص إلى اسود (0) والخلفية إلى أبيض (F)
طباعة نص (سيظهر باللون الأسود)
جعل البرنامج في حالة انتظار

ملاحظات:

١. يتم تمثيل الألوان برقم "سادس عشري" من صفر إلى f حيث يمثل جميع الألوان الأساسية.
٢. عند كتابة رقم واحد "color 9" فهذا سيغير لون النص فقط.
٣. عند كتابة رقمين "color f0" فإن الأول سيغير لون النص والثاني سيغير لون الخلفية.
٤. عند كتابة رقمين متشابهين "color 99" فلن يتغير أي لون، باعتبار أن لون الخط سيشبه لون الخلفية ولذا لن يظهر شيء فلذلك يتم تجاهل الألوان وإعادة الألوان الافتراضية.

تنسيق الألوان الافتراضي لمحرر بيئة Microsoft C++:

١. الكلمة المحجوزة تظهر بلون أزرق.
٢. الكلمة غير المحجوزة تظهر بلون أسود.
٣. التعليقات تظهر بلون أخضر.
٤. لغة C++ حساسة لحالة الأحرف (r لا يساوي R)



التنسيق باستخدام (cout):

تحتوي cout على العديد من الخواص الخاصة بالتنسيق ومنها :

- width والتي تعمل إزاحة لليمين من جهة اليسار بمقدار معين
- Fill والتي تقوم بتعبئة الفراغات التي تركتها width برمز معين.
- Precision والتي تقوم بتقريب الأرقام.

```
#include <iostream.h>
main(){
 cout.
}
```


مثال ١:

```
1. cout.width(20);

2. cout << "Welcome";
```

النتيجة:

```
             Welcome
```

مثال ٢:

```
1. cout.width(20);
2. cout.fill('#');
3. cout << "Welcome";
```

النتيجة:

```
#############Welcome
```

مثال ٣:

الكود:

```
1.  cout.precision(3);
2.  cout << 3.449;
3.  cout.precision(2);
4.  cout << 3.449;
5.  cout.precision(3);
6.  cout << 3.482;
7.  cout.precision(2);
8.  cout << 3.482;
9.  cout.precision(1);
10. cout << 3.482;
11. cout.precision();
12. cout << 3.482;
```

النتيجة:

```
3.45
3.4
3.48
3.5
3
3
```

الفصل 7

العمليات في ++C
C++ Operations

العمليات الحسابية:

الرمز	التوضيح
+ - * /	العمليات الرياضية
y / x	القسمة
y \ x	القسمة الصحيحة
y % x	باقي القسمة

مثال :

```
1.   int x = 3 ;
2.   int z = 7 ;
3.
4.   cout <<  "z + x= " ;
5.   cout << z + x ;
```

الناتج هو " =z + x"

الناتج هو 10

ملاحظة : ما داخل الأقواس 'المفردة' أو "المزدوجة" يعتبر نص.

عمليات المقارنة:

الرمز	التوضيح
>	أكبر من
<	أصغر من
> =	أكبر أو يساوي
< =	أصغر أو يساوي
= =	يساوي
! =	لا يساوي

مثال :

```
1.  int x = 3 ;
2.  int z = 7 ;
3.
4.  if ( x != z)
5.  {
6.     cout << "Not equal" ;
7.  }
```

إذا كان[1] x لا يساوي[2] z

فاطبع الجملة "Not equal"

ناتج البرنامج "Not equal" لأن ناتج الشرط "صواب" True

(1) جمل التحكم في الفصل الثامن.

(2) المساواة تعني "مقارنة قيمتين"، الأمثلة في الصفحة التالية.

العمليات المنطقية Logic Effects:

الرمز	التوضيح
&&	و
||	أو
!	نفي

مثال :

```
1. int x = 3 ;
2. int z = 7 ;
3.
4. if ( x >0 &&  z>0)
5. {
6.    cout << "Both numbers positive" ;
7. }
```

إذا كان x أكبر من الصفر وأيضاً z أكبر من الصفر (ناتج الشرط True)

ناتج البرنامج " Both numbers positive"

المساواة والإسناد Equal and Assigned:

الإسناد: هو إعطاء المتغير قيمة:

مثال:

```
1 int x  ;
2 x = 7 ;

3 cout << x;
```

الناتج 7

المساواة: هو مقارنة قيمتين:

مثال:

```
1. int x = 7  ;
2. int y = 7 ;
3.
4. cout << x==y;
5. cout << x > 3
6.
7. int z = x == 8
8. cout << z ;
```

الناتج صواب (True) "سيطبع 1 في الشاشة"

الناتج خطأ (False) "سيطبع 0 في الشاشة"

اسناد ناتج المقارنة للمتغير z

الناتج False لأن x تساوي 7 وليس 8

وستتم طباعة 0 على الشاشة.

ملاحظة:

x = = y تعني (هل أن x يساوي y) وهي عملية مقارنة ناتجها إما صواب أو خطأ.

تحويل المعادلات الرياضية إلى معادلات برمجية:

	المعادلة الرياضية	المعادلة البرمجية
1)	$z = x^2 + x + 7$	Z = pow(x , 2) + x + 7 ;
2)	$z = \frac{x+1}{y+1}$	Z = (x+1) / (y+1) ;
3)	$z = \frac{(x^2 + x + 7)^2}{y + x + 1}$	Z = pow((pow(x , 2) + x + 7) , 2) / (y + x + 1)

توجد الدالة pow ضمن المكتبة math.h لذلك يجب تضمين المكتبة math.h في البرنامج، المزيد من الدوال الرياضية في الفصل الخامس .

س: كيف تكتب المعادلة التالية برمجياً؟

$$y = \begin{cases} x + 1 : x < 0 \\ x^2 + x + 7 : x > 0 \end{cases}$$

ج:

إذا كان x أصغر من الصفر

وإذا كان غير ذلك[1] (جميع الحالات التي لا توجد في الشرط السابق مثل x>0 أو x=0)

```
1. if ( x < 0 )
2. {
3.     Y = x + 1 ;
4. }
5. else
6. {
7.     Y = pow( x , 2 )  + x + 1 ;
8. }
```

س: كيف تكتب المعادلة التالية برمجياً؟

$$R = x^{y^2}$$

ج:

```
R = pow( x, pow( y , 2 ) ) ;
```

س: كيف تكتب المعادلة التالية برمجياً؟

$$Y = \sqrt{3^2}$$

ج:

```
Y = sqrt( pow( 3 , 2 ) ) ;
```

أولوية العمليات الحسابية:

٦) ما بداخل الأقواس.
١- الأس.
٢- الضرب ثم القسمة.
٣- الجمع أو الطرح.

(1) جملة التحكم if لها عدة أشكال المزيد في الفصل الثامن.

برنامج لإيجاد مساحة المستطيل:

من المهم :
- فهم فكرة البرنامج.
- تحويل الفكرة إلى خطوات منطقية، وأهمها معرفة معادلة مساحة المستطيل.

مساحة المستطيل = الطول × العرض

إذاً نحتاج إلى ٣ متغيرات ، متغيرين سيقوم المستخدم بإدخالهما (الطول والعرض) ومتغير سيحتوي على ناتج ضرب المتغيرين (المساحة).

خطوات الحل البرمجي:

١) التصريحات Declare (تعريف المتغيرات).
٢) المدخلات Input (إسناد قيم للمتغيرات "الطول والعرض").
٣) المعالجة Process (الضرب).
٤) المخرجات Output (مساحة المستطيل).

```
1. int height, width, area;
2. cin >> height >> width;
3. area = height * width
4. cout << area;
5.
```

تعريف ثلاثة متغيرات من نفس النوع في سطر واحد :
استقبال قيمتين من المستخدم:
تخزين نتيجة ضرب القيمتين في متغير المساحة
عرض الناتج (مساحة المستطيل)
الناتج:

```
9
3
27Press any key to continue_
```

كما نلاحظ أن الناتج في المثال السابق غير واضح وعند تنفيذ البرنامج تظهر شاشة سوداء لا يوجد فيها أي معلومات تساعد مستخدم البرنامج في معرفة ما المطلوب منه وماذا يجب أن يعمل..

تنسيق المخرجات:

أعادة المثال السابق:

```
1. int height, width, area;
2. cout << "Enter Height: ";
3. cin >> height;
4. cout << "Enter Width: ";
5. cin >> width
6. area = height * width
7. cout << "--------\n";
8. cout << "Resutl is: " <<area << endl;
```

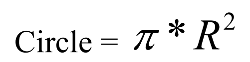
طباعة نص يطلب إدخال الطول
استقبال الطول من المستخدم
طباعة نص يطلب إدخال العرض
استقبال العرض من المستخدم

طباعة خط أفقي وسطر جديد
طباعة "النتيجة هي" ثم ناتج الضرب ثم سطر جديد
الناتج:

```
Enter Height: 7
Enter Width: 8
---------
Result is: 56
Press any key to continue_
```

الواجب:

أكتب برنامج لحساب مساحة الدائرة إذا علمت أن:



$$Circle = \pi * R^2$$

حيث أن :
R : نصف القطر (معطى).
π : 3.14 (ثابت).



الفرق بين signed و unsigned:

signed جعل نوع البيانات يقبل القيم السالبة وهو الافتراضي حتى إذا لم يكتب، بينما unsigned لا يقبل القيم السالبة ، حيث يرفض أي قيمة سالبة وتظل قيمته عشوائية إلى أن يتم إسناد قيمة موجبه إليه.

مثال:

		الناتج
1.	#include <iostream.h>	
2.		
3.	main ()	
4.	{	
5.	signed int x;	
6.	unsigned int z;	
7.	x = 100;	
8.	z = 100;	
9.	cout << x << endl;	100
10.	cout << y << endl;	100
11.	x = -100;	
12.	z = -100;	
13.	cout << x << endl;	-100
14.	cout << y << endl;	4294967196
15.	}	

والرقم 4294967196 عبارة عن رقم عشوائي جاء من الذاكرة نتيجة لأن المتغير y لم يسند له أي قيمة، وذلك لأن النوع unsigned يجعل المتغير يرفض أن يسند إليه قيمة سالبة.

الفصل

8

جمل التحكم
Control Statements

الجمل الشرطية Condition Statements:

١) جملة IF:

```
if ( Expression )
{
      Statements;
      Statements;
      .....;
}
```

شرط
No
Yes
الخروج من if
تنفيذ ما داخل if

```
if ( Expression )
{
      Statements;
      Statements;
      .....;
}
else
{
      Statements;
      Statements;
      .....;

}
```

شرط
No
Yes
تنفيذ ما بداخل Else
تنفيذ ما داخل if

```
if ( Expression ) {
      Statements;
      Statements;
      .....;
}
else if ( Expression ) {
      Statements;
      Statements;
      .....;

}
else {
      Statements;
      Statements;
      .....;

}
```

شرط
No
Yes
تنفيذ ما داخل if 1
شرط
No
Yes
تنفيذ ما داخل if 2
شرط
No
Yes
تنفيذ Else الأخيرة
تنفيذ ما داخل if 3

٢) جملة switch:

```
switch ( var )
{
     case 1 :
          Statements1;
     break;

     case 2:
          Statements;
     break;

     default:
          Statements;
}
```

test
constant 1
statement
break
constant 2
statement
break
default
statement
break

ملاحظة :
فائدة break بعد كل case أنها توقف عمل switch بعد تنفيذ جملة case المناسبة، وإذا لم يتم كتابة الكلمة break فإن البرنامج سينتقل إلى case التالية وينفذها حتى لو لم ينطبق الشرط var عليها.
يمكن كتابة الجزء الأخير الخاص بـ default ويمكن عدم كتابته، حيث يتم تنفيذ الجمل داخل default عندما لا تتحقق أي من الحالات "cases" السابقة، فهي تشبه else الأخيرة في جملة if.

مثال:

```
1.  int r = 3
2.  switch ( var )
3.  {
4.       case 1 : cout << "one\n";     break;
5.       case 2:  cout << "two\n";     break;
6.       case 3:  cout << "three\n";     break;
7.       default: cout << "Error!\n";
8.  }
```

ملاحظة:
إذا كان المتغير حرفي نستخدم علامة الاقتباس المفردة مثل: 'case 'y وإذا كان نصي نستخدم علامة الاقتباس المزدوجة مثل: "case "yes.

الواجب :

اكتب برنامج لمعرفة نوعية العدد (سالب أو موجب أو غير ذلك).

دوال الدوران Loops Functions:

١) For:

تحتاج دالة for إلى عداد (رقم تبدأ منه الدوران ورقم تنتهي إليه) لكي تنفذ الدورات ومقدار الزيادة[1].

الشكل العام Public formula:

```
for ( initialization_value; condition; Increment or Decrement)
    Statements...
```

مثال :

الكود:	النتيجة:
1. for (int i = 0; i <=3; i++)	i value is: 0
2. {	i value is: 1
3. cout << "i value is : " << i << endl;	i value is: 2
4. }	i value is: 3

٢) While:

تحتاج دالة while إلى شرط يحدد استمرارها أو توقفها، فهي ستستمر بلا توقف طالما الشرط متحقق.

الشكل العام Public formula:

```
while (condition)
    Statements...
```

مثال :

الكود:	النتيجة:
1. int w = 3;	
2. while (w <=3)	value is: 0
3. {	value is: 1
4. cout << "value is : " << w << endl;	value is: 2
5. w++;	value is: 3
6. }	

ملاحظة: يمكن الاستغناء عن الأقواس {} الخاصة بدالة for و while و if إذا كانت الجملة التي تنفذها تتكون من سطر واحد.

(1) مقدار الزيادة أو النقصان في الصفات القادمة.

٣) do while:

الشكل العام Public formula:

```
do
{
    Statements…
}
while (condition)
```

مثال :

النتيجة:	الكود:
value is: 0 value is: 1 value is: 2 value is: 3	1. int w = 0; 2. do 3. { 4. cout << "value is : " << w << endl; 5. w++; 6. } 7. while (w <=3);

لكن do while تقوم بتنفيذ الكود مرة واحدة حتى لو كان الشرط خاطئاً:

النتيجة:	الكود:
value is: 0	1. int w = 0; 2. do 3. { 4. cout << "value is : " << w << endl; 5. } 6. while (w >0);

القيمة التزايدية Increment value (معنى ++i):

تعني زيادة المتغير i بمقدار واحد فقط وهو اختصار للجملة التالية:

```
i = i + 1;
```

ويمكن زيادة المتغير i بأي مقدار نريد بالشكل التالي:

```
i += 5;
```

وهذا يكافئ السطر التالي:

```
i = i + 5;
```

أو إنقاص المتغير i بأي مقدار نريد بالشكل التالي:

```
i -= 5;
```

وهذا يكافئ السطر التالي:

```
i = i - 5;
```

الفرق بين ++i و i++ :

مثال ١:

طباعة i ثم زيادته بمقدار واحد.	`cout << i ++;`
زيادة i بمقدار واحد ثم طباعته.	`cout << ++ i;`

مثال ٢:

`int z , i ;` `i = 1 ;` `z = 5 * i + ++ i ;`	`int z , i ;` `i = 1 ;` `z = 5 * i + i ++ ;`
التنفيذ :	التنفيذ :
`z = 5 * 1 + 2;`	`z = 5 * 1 + 1;`
قيمة z:	قيمة z:
7	6

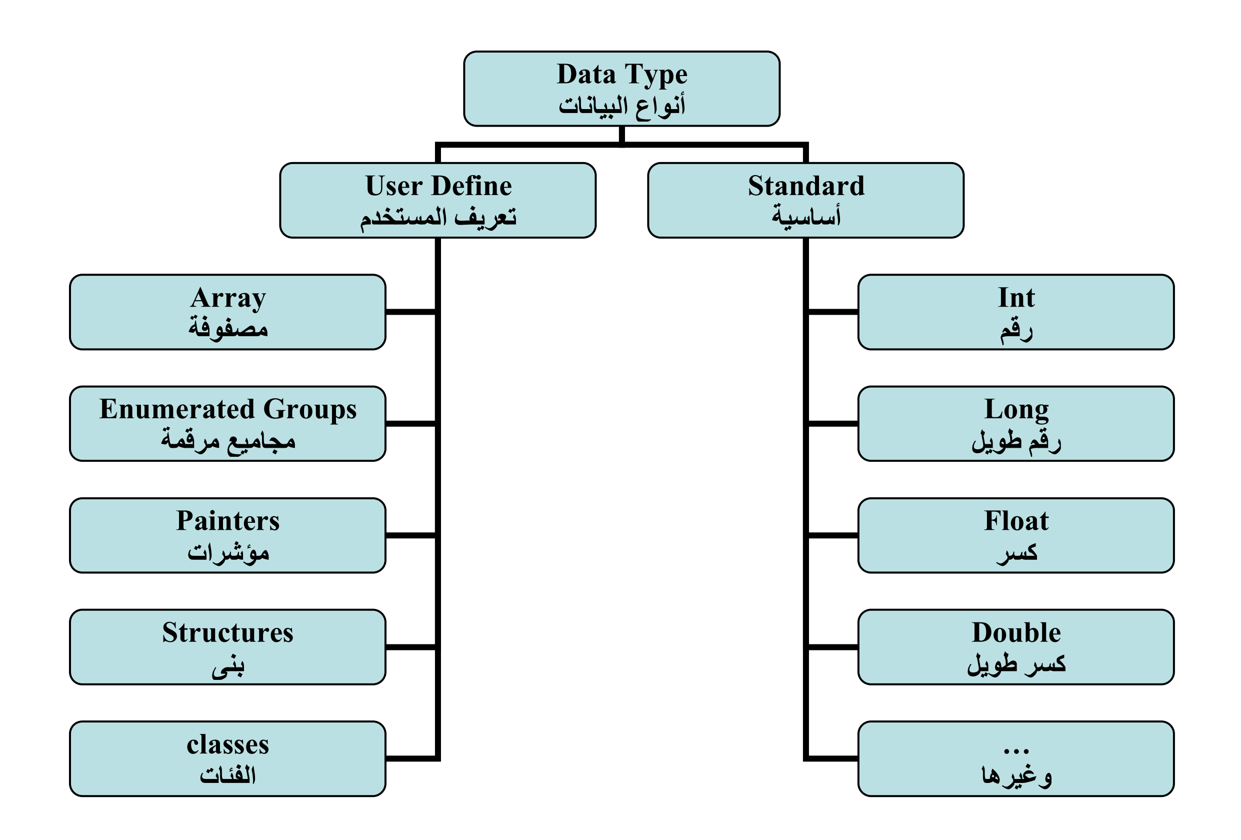
الواجب:
عمل برنامج يقم بطباعة مثلثات باسكال باستخدام دالتي for فقط.

```
        ١
      ٢   ٢
    ٣   ٣   ٣
  ٤   ٤   ٤   ٤
٥   ٥   ٥   ٥   ٥
```

للتسهيل يمكن عملها باستخدام رمز النجمة (*) فقط:

```
        *
      *   *
    *   *   *
  *   *   *   *
*   *   *   *   *
```

أنواع البيانات
Data Types



الأنواع القياسية Standard:
هي أنواع بيانات معرفة مسبقاً يمكن استخدامها ولا يمكن تغييرها فلا يمكن زيادة حجمها أو تحديدها، أي لا يمكن للمستخدم التحكم بها.

الأنواع من تعريف المستخدم User Define:
هي أنواع بيانات يعرفها المستخدم بالطريقة التي يريد ويستطيع تغييرها والتحكم بمساحتها.

المجاميع المرقمة Enumerated groups:

int يعتبر من المجاميع المرقمة المعرفة مسبقاً، ويمكننا تعريف مجاميع خاصة بنا حسب الحاجة.

أمثله[1]:

int نوع بيانات معرف مسبقاً ولا يمكن تغييره.	enum **int** {-32,700, ……. , 32.700}
months معرف من المستخدم ويمكن تغييره أو حذفه.	enum **months** { jan, feb, … , des}

أمثله:

	1. enum **weekday** { sat, sun, …., friday}
تعريف x من نوع weekday	2. weekday **x**;
x لا يقبل إلا قيمة من نوع weekday	3. **x** = sat ;
الناتج: 0	4. cout << **x**;
	5. **x** = sun ;
الناتج: 1	6. cout << **x**;

إعادة تعريف البيانات Data definition return :

أنواع البيانات مثل int, float, double يمكن إعطائها مسميات أخرى للتبسيط أو للحماية حيث إذا تم تغيير نوع البيانات فلن يعرف من يطلع على الكود ما هو هذا النوع.
يمكن تغيير اسم نوع البيانات من خلال الدالة TypeOf مع ملاحظة أن هذا التغيير لا يؤثر على نوع البيانات الأصلي.

إطلاق اسم جديد على النوع float (الأعداد الكسرية).	1. TypeOf float **fl** ;
إسناد قيمة للنوع الجديد (أصبح fl يقوم بعمل float)	2. **fl** x = 5.4 ;
الناتج : 5.4	3. cout << x;

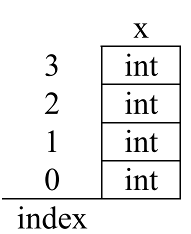
(1) تم التعويض بالنقاط (sun, …., friday) لاختصار المثال.

المصفوفات Array:

جاءت المصفوفات لحل مشكلة الحاجة لإدخال عدد كبير من البيانات، فبدلاً من استخدام عدد كبير من المتغيرات لحفظ البيانات يتم استخدام المصفوفة التي تستطيع الاحتفاظ بالبيانات كمتغير واحد.

تعريفها:

مجموعة من المواقع المتجاورة في الذاكرة ولها نفس نوع البيانات وتستخدم لخزن البيانات.



أنواع المصفوفات Array types:

١. مصفوفات أحادية البعد Single Dimensional.
٢. مصفوفات متعددة الأبعاد Multi Dimensional.

مميزات المصفوفات:

١) تقليل حجم البرنامج.
٢) سهولة اسناد القيم واسترجاعها.
٣) استخدام تقنيات البحث والترتيب.
٤) الوصول المباشر Direct Access إلى البيان، وهذه ميزة غير موجودة في أي نوع آخر.

عيوب المصفوفات:

١) يمكن للمستخدم تحديد حجم المصفوفة عند تعريفها فقط (لا يمكن أثناء التشغيل تحديد حجمها).
٢) يجب أن تحتوي جميع القيم على نوع واحد من البيانات (لا يمكن تخزين بيانات من أنواع مختلفة).

الصيغة العامة Public formula:

١. مصفوفة أحادية:

```
Data_Type Array_name [ Array_Size ];
```

مثال:

```
int x[5];
```

حجم المصفوفة يتحدد بنوعها فإذا كانت رقمية int فإن كل عنصر فيها سيحجز 4 بايت، وبالتالي فإن حجم المصفوفة الأحادية سيكون بضرب حجم نوع البيانات في عدد صفوف المصفوفة كالتالي:

x
4 byte
4 byte
4 byte
4 byte
4 byte

$$4 \text{ byte} * 5 \text{ rows} = 20 \text{ bytes}$$

وهذه هي المساحة المحجوزة للمغير x في الذاكرة

٢. مصفوفة ثنائية:

```
Data_Type Array_name [ Array_Rows_Size ][ Array_Cols_Size ];
```

مثال:

```
int x[3][4];
```

x

4 byte	4 byte	4 byte	4 byte	4 byte
4 byte	4 byte	4 byte	4 byte	4 byte
4 byte	4 byte	4 byte	4 byte	4 byte

حجم المصفوفة متعددة الأبعاد يكون بضرب حجم نوع البيانات × عدد صفوف المصفوفة × عدد أعمدة المصفوفة:

4 byte * (3 rows * 4 cols) = 48 bytes

إدخال البيانات للمصفوفة (الإسناد) [1]:

الطريقة الأولى:

تعريف مصفوفة وإسناد كل بياناتها في نفس الوقت:
بقية الخانات ستكون صفرية:
كل الخانات ستكون أصفار:

```
1. int x[5] = { 1, 7, 10, 2, 5};
2. int y[10] = {3, 5};
3. int z[4] = {0};
```

الطريقة الثانية:

تعريف مصفوفة:
إسناد قيمة للخانة الأولى في المصفوفة:
إسناد قيمة للخانة الثانية في المصفوفة:
إسناد قيمة للخانة الثالثة في المصفوفة:

```
1. int a[3];
2. a[0] = 1;
3. a[1] = 7;
4. a[2] = 10;
```

10
7
1

ويمكن إدخال البيانات إلى المصفوفة أثناء تشغيل البرنامج عن طريق (cin) :

إدخال جميع قيم المصفوفة باستخدام دالة For

```
1. int a[2];
2. for ( int i=0; i<=2; i++)
3. {
4.     cin >> a[i];
5. }
```

وبالمثل عملية الإخراج:

طباعة جميع قيم المصفوفة باستخدام دالة For

```
1. for ( int i=0; i<=2; i++)
2. {
3.     cout << a[i] << endl;
4. }
```

(1) عند تحديد حجم المصفوفة بخمسة في لغة C++ فإن عدد العناصر خمسة أما في لغة Visual Basic فإن عدد العناصر سيكون ستة.

إدخال بيانات من النوع الحرفي char:

النوع char يقبل حرف واحد فقط ويحجز بايتاً واحداً في الذاكرة

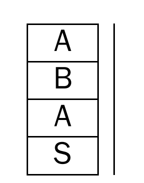
```
char c;
char c = 'a';
```

تعريف وإسناد معاً

كلا الطريقتين الآتيتين تؤدي نفس الغرض في الإدخال إلى المصفوفة:

- إسناد كل حرف على حدة
- إسناد كل الحروف دفعة واحدة.

```
char x[4] = { 'S', 'A', 'B', 'A'};
char x[4] = "SABA";
```



ومن هنا نستنتج أن النوع String ما هو إلا مصفوفة حرفية وحجمها 255 بايتاً.

النوع string يستخدم نفس طريقة الإسناد.

```
string = "SABA";
```

ملاحظة :
عند تعريف وإسناد المصفوفة في نفس الوقت فإن لغة C++ تحدد حجم المصفوفة تلقائياً بمعرفة العناصر داخل الأقواس. أما إذا كنت ستعرف المصفوفة وتسند القيم فيما بعد فتأكد أن القيمة داخل قوسي المصفوفة [ ] تساوي عدد عناصر المصفوفة.

مثال:

```
int x[] = {1,2,3,4};
```

استخدام المصفوفة:

جمع قيمتين في المصفوفة وإسناد الإجمالي إلى متغير آخر:

```
1. int z;
2. int x[5] = { 1, 7, 10, 2, 5};
3. z = x[0] + x[3];
```

جمع جميع قيم عناصر المصفوفة:

```
1. int sum = 0;
2. for ( int i =0; i<5; i++)
3. {
4.     sum = sum + x[i];
5. }
6. cout << sum;
```

ضرب جميع قيم عناصر المصفوفة:

```
1. int m = 1;
2. for ( int i =0; i<5; i++)
3. {
4.     m = m * x[i];
5. }
6. cout << m;
```

المصفوفة متعددة الأبعاد:

- ثنائية البعد:

```
Data_Type Array_name [ x ][ y ];
```

- ثلاثية البعد:

```
Data_Type Array_name [ x ][ y ][ z ];
```

- رباعية البعد:

```
Data_Type Array_name name [ x ][ y ][ z ][ t ];
```

وهكذا يمكن إضافة أبعاد بحسب الحاجة.

مثال:

```
char ary[3][3] = {{'A','B','C'},{'D','E','F'},{'G','H','I'}};
```

A	B	C
D	E	F
G	H	I

إدخال البيانات:

كلا الثلاثة الطرق الآتية تؤدي نفس الغرض في الإدخال إلى المصفوفة الثنائية:

- الطريقة الأولى:

```
1.  int x[2][2] = {1, 5, 4, 9};
```

- الطريقة الثانية:

```
1.  int x[2][2] = { {1, 5} , {4, 9} };
```

- الطريقة الثالثة:

```
1.  int x[0][0] = 1;
2.  int x[0][1] = 5;
3.  int x[1][0] = 4;
4.  int x[1][1] = 9;
```

يمكن إدخال بيانات المصفوفة الثنائية باستخدام دالتي For:

```
1.  int x[2][2];
2.
3.  for ( int i =0; i<2; i++)
4.      for ( int r =0; r<2; r++)
5.          cin >> x[i][r];
```

وكذلك الإخراج:

```
1.  for ( int i =0; i<2; i++)
2.      for ( int r =0; r<2; r++)
3.          cout << x[i][r];
```

العمليات على المصفوفات:

- الإضافة.
- الحذف.
- البحث.
- الترتيب.
- التعديل
- وغيرها...

أ) البحث:

١- البحث عن قيمة موجودة في المصفوفة:

يتم البحث عن قيمة موجودة في المصفوفة بمقارنة القيمة المراد البحث عنها في عناصر المصفوفة.

مثال:

```
1.  int x = 4;
2.  int a[] = {1,2,3,4,5,6};
3.  int t;
4.  for ( int i =0; i<6; i++) {
5.      if ( x == a[i]) {
6.          t = 1;
7.          break;

8.      } else {
9.          t = 0;
10.     }
11. }

12. if ( t = 1)
13.     cout << "number found\n";
14. Else
15.     cout << "number not found\n";
```

المرور على كل عناصر المصفوفة بدالة for
إذا تساوت قيمة x مع قيمة في المصفوفة
فيتم اسناد قيمة 1 للمتغير t
وكذلك الخروج من دالة for الى السطر رقم 12
باستخدام الكلمة Break.
وإذا لم تتساوى قيمة x مع قيمة في المصفوفة
فيتم إسناد قيمة 0 للمتغير t

* ستدور الدالة for أربع مرات حتى تجد العنصر الذي قيمته 4، وعندها فإن السطر 6 سينفذ وسيكون t=1 وستقفز إلى السطر رقم 12 بواسطة الأمر Break وعندها سيتم طباعة النص في السطر 13
وإذا لم تجده "نفرض أننا نبحث عن القيمة 8" فسيتم طباعة النص في السطر 15

٢- البحث عن أكبر قيمة موجودة في المصفوفة:

يتم البحث بأخذ عنصر من المصفوفة ومقارنته ببقية العناصر وخزن القيمة الكبيرة في temp .

```
1.  int a[11] = {5,3,7,10,8,6,2,4,11,2,0};
2.  int temp=0;
3.  for (int j=0; j<11; j++){
4.      if ( a[j] > temp ){
5.          temp = a[j];
6.      }
7.  }
8.  cout << temp << endl;
```

ب) الترتيب:

```
1.  int a[] = {5,3,7,10,8,6,2,4,11,2,0};
2.  int temp = 0;
3.  for (int i =0; i< 11; i++) {
4.      cout << temp << "\t [" << ends ;
5.      for (int r=0; r< 11; r++) {
6.          if(a[r+1]<a[r]){
7.              temp = a[r];
8.              a[r] = a[r+1];
9.              a[r+1] = temp;
10.         }
11.         cout << a[r] << ends;
12.     }
13.     cout << "]" <<endl;
14. }
15. cout << "===============\n";

16. for (j=0; j<11; j++){
17.     cout << a[j] << ends;
18. }
19. cout << endl;
```

السطر 4: طباعة قيمة temp لغرض مشاهدة التغييرات

السطور 6-10: إذا كان العنصر التالي أكبر من السابق بدل المواقع السطور 6,7,8 عبارة عن تبديل مواقع القيم في خلايا المصفوفة . وإذا كان العنصر التالي أصغر من السابق فلن يتم تبديل موقعه حتى لو لم يكن أصغر قيمة في المصفوفة فسينتظر لحين الدورة الثانية لـ For الأولى التي ستقوم بإعادة الترتيب وتحسين النتائج أكثر وأكثر، انظر موقع القيمة 0 في الصورة أدناه.

السطر 11: طباعة المتغير لغرض مشاهدة التغييرات

السطور 16-18: طباعة المصفوفة بعد الترتيب وهو ما يهمنا

```
0       [ 3 5 7 8 6 2 4 10 2 0 11 ]
11      [ 3 5 7 6 2 4 8 2 0 10 11 ]
10      [ 3 5 6 2 4 7 2 0 8 10 11 ]
8       [ 3 5 2 4 6 2 0 7 8 10 11 ]
7       [ 3 2 4 5 2 0 6 7 8 10 11 ]
6       [ 2 3 4 2 0 5 6 7 8 10 11 ]
5       [ 2 3 2 0 4 5 6 7 8 10 11 ]
4       [ 2 2 0 3 4 5 6 7 8 10 11 ]
3       [ 2 0 2 3 4 5 6 7 8 10 11 ]
2       [ 0 2 2 3 4 5 6 7 8 10 11 ]
2       [ 0 2 2 3 4 5 6 7 8 10 11 ]
==================
0 2 2 3 4 5 6 7 8 10 11
Press any key to continue_
```

حل آخر:

```
1.  int a[] = {5,3,7,10,8,6,2,4,11,2,0};
2.  int temp = 0;
3.  for (int i =0; i< 11; i++) {
4.      cout << temp << "\t [" << ends ;

5.      for (int r=i+1; r< 11; r++) {
6.          if(a[i]>a[r]){
7.              temp = a[r];
8.              a[r] = a[i];
9.              a[i] = temp;
10.         }
11.         cout << a[r] << ends;
12.     }
13.     cout << "]" <<endl;
14. }
15. cout << "===============\n";

16. for (int j=0; j<11; j++){
17.     cout << a[j] << ends;
18. }
19. cout << endl;
```

```
0         [ 5 7 10 8 6 3 4 11 2 2 ]
0         [ 7 10 8 6 5 4 11 3 2 ]
2         [ 10 8 7 6 5 11 4 3 ]
2         [ 10 8 7 6 11 5 4 ]
3         [ 10 8 7 11 6 5 ]
4         [ 10 8 11 7 6 ]
5         [ 10 11 8 7 ]
6         [ 11 10 8 ]
7         [ 11 10 ]
8         [ 11 ]
===============
0 2 2 3 4 5 6 7 8 10 11
Press any key to continue
```

ج) التعديل:

س: إذا علمت أن المصفوفة التالية تمثل مجموع درجات الطلاب، فالمطلوب إضافة 5 درجات للطلاب الذين تنقصهم 5 درجات للنجاح.

int Stud[6] = {70, 50, 45, 43, 47, 90};

فالمطلوب عمل الكود اللازم.

ج:

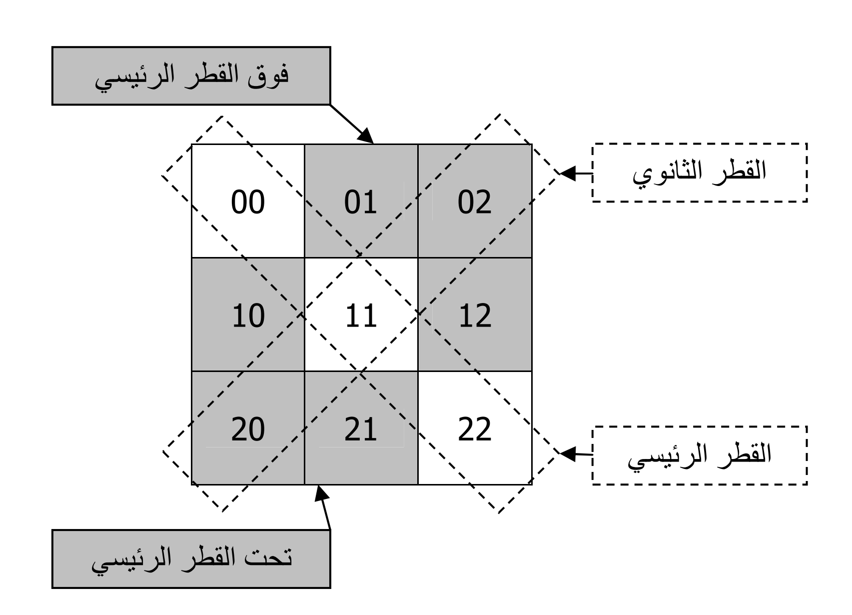
```
1.  for ( int i =0; i<6; i++) {
2.      if (Stud [i] < 50 && Stud[i] > 45) {
3.          // Stud[i] += 5;   // A
3.          // Stud[i] = 50;   // B
4.      }
5.  }
6.  for (  i =0; i<6; i++)
7.      cout << Stud[i] << ", " << ends;
8.  cout << endl;
```

طريقة A: زيادة الطالب 5 درجات.
طريقة B: إعطاء الطالب درجة النجاح فقط.
الغي التعليق من أحد الطريقتين A أو B في في السطور رقم 3 لعرض الناتج بالطريقة التي تختارها.

الناتج (بالطريقة الثانية B):

70, 50, 45, 43, 50, 90

الواجب:



في الشكل أعلاه: المطلوب عمل برنامج يقوم بالتالي:

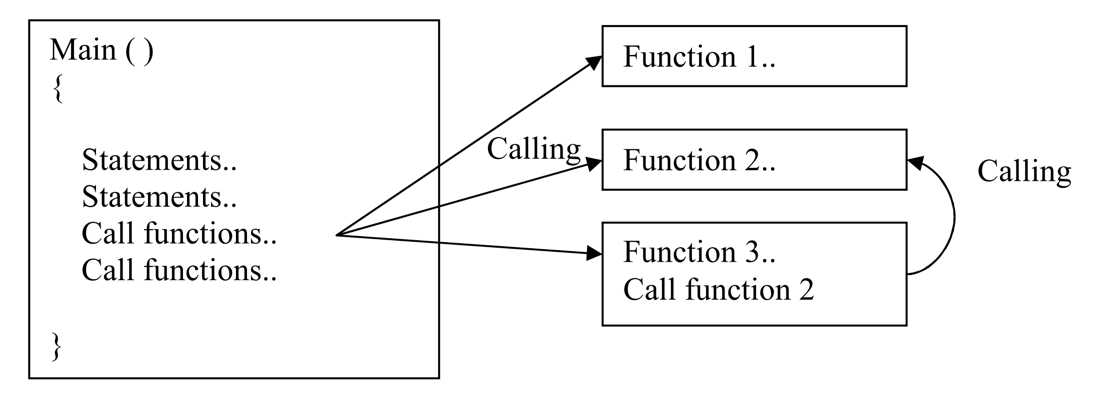
- طباعة عناصر القطر الرئيسي. (مساعدة: يتساوى فيه رقم الصف مع رقم العمود)
- طباعة عناصر القطر الثانوي. (مساعدة: رقم الصف + رقم العمود = حجم المصفوفة - ١)
- طباعة عناصر فوق القطر الرئيسي. (مساعدة: رقم الصف أكبر من رقم العمود)
- طباعة عناصر تحت القطر الرئيسي. (مساعدة: رقم الصف أصغر من رقم العمود)

الدوال Functions:

وهي عبارة عن برامج فرعية تشبه البرنامج الرئيسي main ، البرنامج main هو عبارة عن دالة تتميز بأن بيئة C++ تقوم بتشغيلها ، وتتولى دالة main تشغيل ما بداخلها واستدعاء الدوال الفرعية وتشغيلها.

تعريفها:

- مجموعة من التعليمات والأنشطة المكتوبة داخل برنامج مستقل (فرعي) يتم استدعائها داخل البرنامج الرئيسي.



ملاحظة:
استدعاء الدوال وتفعيلها يجب أن يكون من داخل الدالة الرئيسية main .

مميزات الدوال:

١- تقسيم البرامج الكبيرة إلى أجزاء صغيرة داخل البرنامج "Source file".
٢- اختصار الكثير من السطور في اسم الدالة (ذات الطابع التكراري).
٣- تنظيم البرنامج.
٤- قابلية استخدامها أكثر من مرة.
٥- سهولة التطوير والتعديل واكتشاف الأخطاء.

أنواع الدوال:

- دوال قياسية Standard function:
كل الدوال الموجودة في المكتبات مثل pow() و cout ولا يمكن تعديلها.

- دوال من تعريف المستخدم User define function:
الدوال التي يقوم المستخدم بتكوينها.

أشكال الدوال:

- **إجراء Procedures:**
وهي دالة تقوم بعمل معين وتنفذه أو تطبعه على الشاشة، و تسمى (إجراء) لأنها لا تعيد قيمة.

- **دالة Function:**
وهي دالة تقوم بعمل معين وتعيد قيمة ويمكنها أن تطبع شيء على الشاشة وتعيد قيمة في نفس الوقت، ويمكن إسناد هذه القيمة إلى متغير ثم طباعته على الشاشة.

الصيغة العامة للإجراء:

التصريح

البناء

```
void FunctionName(parameters)
{
    Statements...
{
```

الصيغة العامة للدالة:

- استخدام الكلمة المحجوزة return لإعادة قيمة:

```
DataType FunctionName(parameters)
{
    Statements...
    return value;
{
```

- استخدام نفس اسم الدالة لإعادة قيمة:

```
DataType FunctionName(parameters)
{
    Statements...
    FunctionName = value;
{
```

يمكن أن يحتوي الإجراء أو الدالة على باراميترات Parameters ويمكن ألا تحتوي عليها، فهذا يرجع للمبرمج.

مثال لإجراء:

```
1.  void sum(int a, int b)
2.  {
3.        cout << a + b;
4.  {
```

استدعاء الإجراء داخل البرنامج:

```
sum( 7 , 8 );
```

الناتج: 15

ملاحظات:

- لا يمكن إسناد الإجراء لمتغير فهو لا يعيد أي قيمة.
- لا يمكن إدخال الإجراء ضمن عمليات حسابية.

مثال لدالة:

```
1. int sum(int a, int b)
2. {
3.       Return (a + b);
4. {
```

استدعاء وطباعة الدالة داخل البرنامج "سيتم طباعة ناتج الدالة":

```
cout << sum( 7 , 8 );
```

الناتج: 15

أو استدعاء وإسناد الدالة إلى متغير "سيتم إسناد ناتج الدالة لمتغير":

```
1. int s;
2. s = sum( 7 , 8 );
3. cout << s;
```

الناتج: 15

استدعاء وطباعة الدالة داخل البرنامج وإدخالها ضمن عملية حسابية "سيتم في العلمية الحسابية التعامل مع ناتج الدالة كقيمة":

```
1. cout << sum( 7 , 8 ) + 5;
```

الناتج: 20

استدعاء وإسناد الدالة إلى متغير وإدخالها ضمن عملية حسابية.. "سيتم إضافة ناتج الدالة إلى العملية الحسابية":

```
1. s = sum( 7 , 8 ) + 5;
2. cout << s;
```

الناتج: 20

تقديم نوع البيانات int قبل اسم الدالة (int function_name) يعني أن الدالة تعيد قيمة من النوع int بينما تقديم الكلمة void قبل اسم الدالة (void function_name) يعني أن الدالة لا تعيد أي قيمة من أي نوع.

تعريف الدوال :

- يمكن تعريف الدوال قبل الدالة الرئيسية main (التصريح والبناء معاً).

```
1.    int sum(int a, int b)
2.    {
3.           return a + b;
4.    {
5.
6.    void main()
7.    {
8.          cout << sum( 5 , 4 );
9.    }
```

- يمكن تعريف الدوال تحت الدالة الرئيسية main بشرط التصريح عنها قبل الدالة.

```
1.    int get_sum( int , int );          // التصريح عن الدالة
2.
3.    void main()
4.    {
5.          cout << get_sum( 5 , 4 );    // طباعة الدالة "ناتجها"
6.    }
7.
8.    int get_sum(int a, int b)          // بناء الدالة
9.    {
10.          return a + b;
11.   }
```

تطبيق المثال[1]:

```
#include <iostream.h>

int get_sum( int , int );
/////////////////////////////////
int main(){

    cout << get_sum()
};          int get_sum ( int a=0, int b=0)
/////////////////////////////////
int get_sum( int a=0, int b=0){
    return a + b;
}
```

الواجب:

- قم بعمل مثلث باسكال يقوم برسم عدة أشكال للمثلث باستخدام دالة واحدة تحوي دالة for واحدة، حيث يتم تغيير شكل المثلث بالتلاعب بقيمة متغيرات الدالة فقط.

- حل التمارين الموجودة لدى المصور إلى نهاية الترم.

---

(1) تم استخدام برنامج (Bloodshed Dev-C++) في هذا المثال وبعض الأمثلة في هذا الكتاب.

القيم الابتدائية في الدوال:

تعتبر كنوع من تحديد القيمة الابتدائية لمتغيرات الدالة أو الإجراء، فيمكنك عدم إسناد أي قيمة لمتغيرات الدالة:

مثال :

```
1.  double _div( float a=0, float b=0){
2.  {
3.     if(b==0){
4.        return 0;
5.     }else{
6.        return (a / b);
7.     }
8.  }
9.
10. void main()
11. {
12.     cout << _div() << endl;
13.     cout << _div(5) << endl;
14.     cout << _div(5, 0) << endl;
15.     cout << _div(8, 5) << endl;
16. }
```

تحديد قيم افتراضية لمتغيرات الدالة "أصفار"

إرجاع 0 إذا كان المتغير الثاني صفراً

إرجاع ناتج قسمة العددين إذا كان المتغير الثاني لا يساوي الصفر.

الناتج: 0

الناتج: 0

الناتج: 0

الناتج: 1.6

ملاحظات:

- يمكن جعل إحدى متغيرات الدالة تحمل قيمة ابتدائية والأخرى لا تحمل قيمة ابتدائية فهذا يعني أنه يجب إسناد قيمة لهذا للمتغير الآخر بشرط أن لا تجعل المتغيرات التي تحمل قيمة ابتدائية قبل المتغيرات التي لا تحمل قيم ابتدائية (لماذا؟ "ابحث عن هذا الموضوع").
- تم كتابة div_ تسبقها شرطة لأن هناك دالة من دوال مكتبة iostream.h تحمل الاسم div.

استخدام #Define:

يستخدم هذا الأمر ليخبر المترجم باستبدال سلسلة من الأحرف بالقيمة المجاورة للأمر define فهذا الأمر لا يفحص نوع القيمة فقد تكون قيمة أو معالجة لعملية حسابية أو غيرها كما في الدوال:

١) استخدام الأمر define لتعريف الثوابت:

الشكل العام Public formula:

```
#define Constant  value;
```

مثال :

```
1.   #define MAX  100;
2.   main ( )
3.   {
4.      cout << MAX;
5.   }
```

الناتج: 100

٢) استخدام الأمر define بدلا عن الدوال:

الشكل العام Public formula:

```
#define Function_name (parameters)   Statements...
```

مثال :

```
1.   #define SUM( x , y ) x + y;
2.   main ( )
3.   {
4.      Int z = SUM( 1 , 2 );
5.      cout << z << endl;
6.      cout << SUM(3.5, 7.5);
7.   }
```

الناتج: 3
الناتج: 11

مميزات define:

- لا يحتاج لتعريف نوع البيانات.
- لا يحتاج لتعريف نوع الدوال.
- يمكن إسناد قيمتها إلى متغير بشرط أن يكون المتغير من نفس نوع البيانات المعادة من الدالة.
- يمكن الاستغناء بها عن التحميل الزائد للدوال overload "سيتم دراسته في الفصل الثاني".

عيوب define:

- لا يمكن عمل مجموعة إجراءات "جمل" في سطور متعددة تحت الأمر define لأن المترجم سيتجاهل السطور اللاحقة ويعتبرها خطأً.

ملاحظات:

- نلاحظ أن هذه الطريقة لا تحتاج لتعريف نوع المتغيرات في الدالة SUM ، ويمكننا عند الاستدعاء أن نكتب أي قيم من أي نوع، لكن ما يحدد نوع المتغيرات المرسلة هو نوع العملية وهي "x+y" ففي هذه الحالة سيحدث خطأ عند إرسال قيم حرفية نظراً لأن جملة الدالة تحتوي على جمع ، حيث لا يمكن جمع قيم نصية.
- هذه الطريقة تشبه إلى حد كبير تعريف الثوابت const حيث لا يمكن تغيير القيمة بعد تعريفها.
- يجب أن يتم وضع define قبل الدالة الرئيسية main "في منطقة التصاريح العامة".

الفصل

10

هياكل البيانات
Data Structures

مقدمة:

تمثل الدوال والمصفوفات طرقاً من طرق هيكلة البرمجة، حيث أنها تساعد في تنظيم وتسهيل كتابة البرنامج.
كما تمثل الهياكل طريقة أخرى من طرق هيكلة البرمجة.

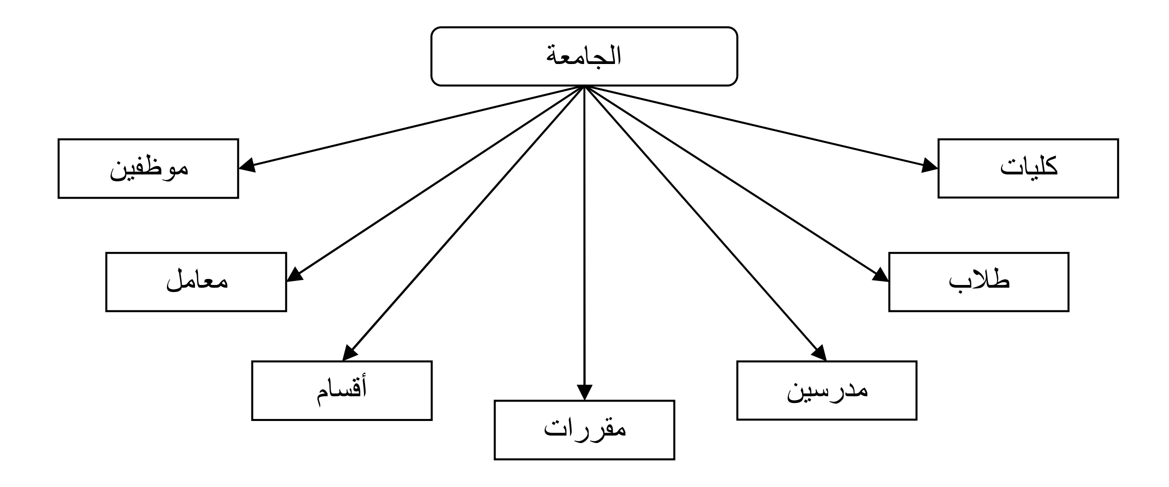
تعريفها:

يطلق عليها هياكل أو تراكيب أو سجلات وهي تعني التعامل مع مجموعة من البيانات كوحدة واحدة .
أو هي مجموعة من المتغيرات والصفات والخصائص "الدوال" التي تندرج تحت بناء واحد "هيكل".

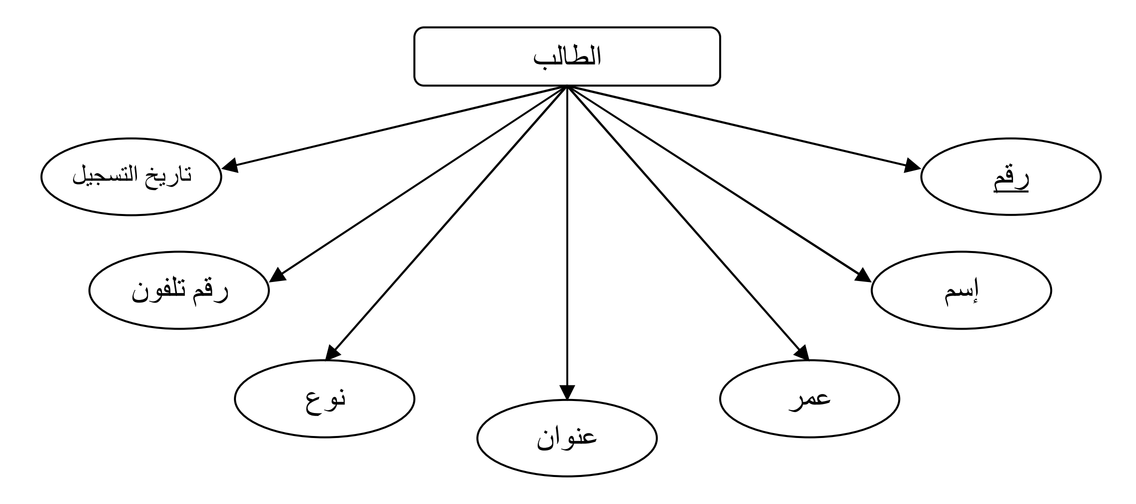
الهيكل التنظيمي:

فكرة هياكل البيانات تشبه الهياكل التنظيمية لأي مؤسسة أو شركة، فمثلاً الجامعة تحتوي على كليات ومقررات ومدرسين وطلاب، لكن كلمة "كلية تقنية المعلومات" أو "مادة ++C" مبهمة إذا ذكرت بدون الإشارة إلى انتمائها فيجب الإشارة إليها والى الجهة التي تتبعها مثل "جامعة سبأ - كلية تقنية المعلومات" أو "جامعة سبأ - تقنية معلومات - مادة ++C".

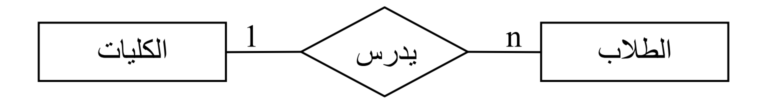
وبالتالي عند عمل أي مشروع برمجي يجب تحويل هيكل المنظمة إلى هيكل بيانات بحسب الاحتياج، فيجب هيكلة ما لا يمكن أن تستمر المنظمة في عملها بدونه وعدم هيكلة الأجزاء غير الأساسية في المنظمة، ولكن هذا ليس شرطاً فللمبرمج الحرية في تحديد ما يجب أن يهيكله وما لا يجب بحسب المشروع الذي ينوي بناءه.
فإذا كان مشروعه نظام جامعة متكامل فمن الضروري أن يحتوي النظام على كل أجزاء الجامعة الرئيسية أما إذا كان برنامج بصمة الكترونية فإنه يحتاج لمعرفة أسماء الموظفين فقط.



وكل جزء من أجزاء المنظمة يحتوي على خصائص فالكلية لديها "اسم" و "أقصى عدد طلاب" و "أقل عدد" والطالب يحتوي على "رقم" و"اسم" و "عمر" و "عنوان" .. الخ.



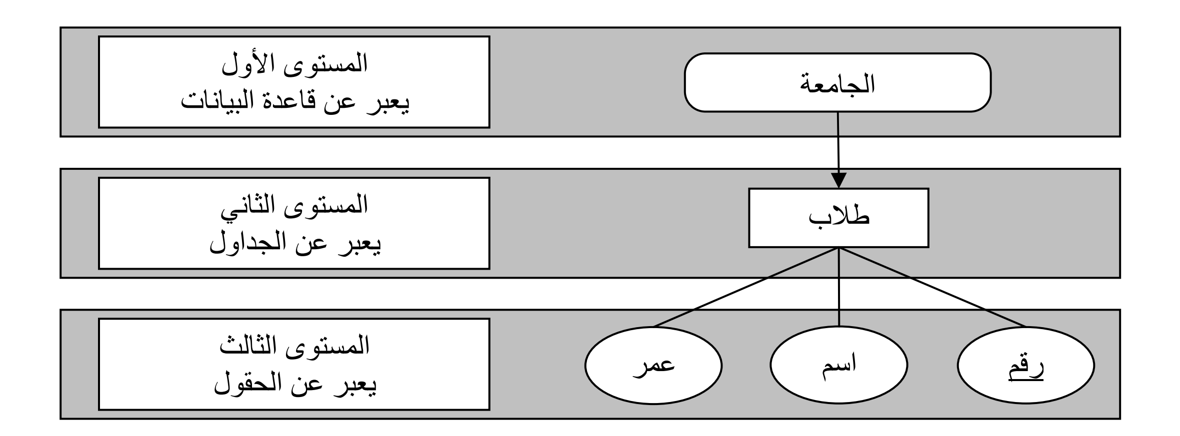
وهناك علاقة بين أجزاء المنظمة فهناك علاقة بين الكليات والطلاب من نوع (واحد الى كثير) فالطالب يدرس في كلية واحدة بينما الكلية يدرس فيها أكثر من طالب (راجع مقرر قواعد البيانات) .



وقد تتطلب العلاقات غير الواضحة أو علاقات كثير لكثير إنشاء كيان وهمي مثل (مقررات الكلية) لربط المقرر بالكلية.

في الحياة الواقعية تفرض طبيعة العمل نفسها على طبيعة العلاقات بين مكونات المنظمة وما يهم المبرمج أن يعكس طبيعة هذه العلاقة في برنامجه بالشكل الأنسب، وهو ليس بهذه السهولة فأي مشروع برمجة يحتاج إلى تحليل وتصميم ثم برمجة، وكل وظيفة من هذه الوظائف تحتاج الى متخصصين:

المحلل : يجمع البيانات ، ويربط المعلومات مع بعض، ولديه خبرة في العلاقات العامة.
المصمم : يأخذ نتائج التحليل ويحولها الى شكل يعبر عن الهيكل ، ولديه خبرة في برامج التصميم.
المبرمج : يحول تصميم الهيكل الى واجهات وأوامر وأكواد.



التركيب Struct:

بعد المقدمة النظرية ننتقل إلى الجزء العملي من الهياكل:

الصيغة العامة Public formula:

```
1.  struct struct_name
2.  {
3.      DataType member1;
4.      DataType member2;
5.      DataType member3;
6.  }
```

ملاحظات:

١. يكتب التركيب قبل الدالة الرئيسية:

مثال لكيان طالب:

```
1.  struct students
2.  {
3.      int num;
4.      char name[20];
5.      int age;
6.      string address;
7.      double phone;
8.  };
9.  void main()
10. {
11.
12. }
```

رقم
اسم
عمر
عنوان
تلفون

٢. يصبح التركيب نوعاً من أنواع البيانات مثله مثل int أو char ... الخ.

Struct:

num	name	age	address	phone
4	20	4	8	4

= 44 byte

٣. يعرّف على التراكيب متغيرات:

```
students stud;
```

إسناد قيم للتركيب:

١. إسناد القيم دفعة واحدة:

```
students stud = {1, " ali ", 25, " sanaa ", 123.456};
```

٢. إسناد كل قيمة بشكل مستقل:

```
1.  students stud;
2.  stud.num = 1;
3.  stud.name = "ali";
4.  stud.age =  25;
5.  stud.address = "sanaa";
6.  stud.phone = 777777777;
```

٣. ويمكن استخدام الدالة cin لطلب البيانات من المستخدم (أثناء التشغيل):

```
1.    students stud;
2.    cout<< "Enter number: ":      cin >> stud.num;
3.    cout<< "Enter name: ":        cin >> stud.name;
4.    cout<< "Enter age: ":         cin >> stud.age;
5.    cout<< "Enter address: ":     cin >> stud.address;
6.    cout<< "Enter phone: ":       cin >> stud.phone;
```

يمكن تعريف نوع التركيب لمتغير بعد بناءه مباشرة:

```
1.    struct notebook
2.    {
3.        int num;
4.        char name[20];
5.        double phone;
6.    } note;
7.    void main()
8.    {
9.         note.num = 1;
10.   }
```

لاحظ المتغير note في السطر رقم (٦).

إسناد قيم للمتغير من نوع notebook

حجم التركيب:

يأخذ التركيب إجمالي حجم أنواع البيانات الداخلة في تركيبه:

حجم أنواع بيانات العادية:

النوع	الحجم
int	4 byte
char	1*20 byte
double	8 byte

حجم التركيب notebook (نوع بيانات مركب):

notebook			الحجم
int	char	double	13 byte

ملاحظة:
جاء التركيب لحل مشكلة المصفوفات في أنه يمكنه احتواء بيانات مختلفة الأنواع.

ملاحظات:

١) في الشكل التالي نلاحظ أن بيئة C++ تعطينا قائمة منسدلة بكل خصائص التركيب مما يسهل علينا كتابة برنامجنا.

```
#include <iostream>
#include <string>

using namespace std;

struct students{

    int num;
    char name[20];
    int age;
    string address;
    double phone;
}

void main(){

    students stud = {1, "ali", 25, "sanaa", 777777777};

    cout << stud.
                 address
}                age
                 name
                 num
                 phone
```

٢) التركيب في الشكل أعلاه لا يستطيع خزن بيانات أكثر من طالب واحد، فعند إدخال بيانات طالب ثاني سيتم حذف بيانات الطالب السابق، وهذا يشبه إسناد القيم للمتغير، فكلما تسند قيمة جديدة للمتغير فإن القيم السابقة تحذف، ولحل هذه المشكلة نستخدم المصفوفات.

استخدام المصفوفات في التراكيب:

البرنامج التالي عبارة عن نظام لتسجيل بيانات الطلاب وعرضها بشكل منسق:

```
#include <iostream>
#include <string>

using namespace std;

struct students{
        int num;
        char name[20];
        int age;
        string address;
        double phone;
};

void main(){
        students   stud[10];          // تعريف مصفوفة من نوع التركيب "Stud"

        for(int i=0; i<3; i++){
                stud[i].num = i;
                cout << "Student number: " << i+1 << endl;
                cout << "---------------------\n\n";

                cout << "Name: ";           cin >> stud[i].name;
                cout << "Age: ";            cin >> stud[i].age;
                cout << "Address: ";        cin >> stud[i].address;
                cout << "Phone: ";          cin >> stud[i].phone;
                system("cls");
        }

        cout << "Num\tName\tAge\tAddress\tPhone\n";
        cout << "------------------------------\n";

        for(i=0; i<3; i++){
                cout << i+1 << "\t";
                cout << stud[i].name << "\t";
                cout << stud[i].age    << "\t";
                cout << stud[i].address << "\t";
                cout << stud[i].phone   << "\n";
        }
}
```

مخرجات الشاشة:

```
Num     Name    Age     Address Phone
------------------------------
1       ali     21      sana'a  177
2       bader   22      aden    277
3       saeed   23      amran   377
Press any key to continue
```

التركيب المتداخل:

هو عبارة عن تركيب داخل تركيب. فقد نحتاج في برنامجنا السابق لتاريخ ميلاد الطالب ولكن لا يوجد في لغة C++ نوع بيانات للتاريخ، لذلك سنقوم بعمل تركيب لتعريف نوع بيانات جديد يمثل التاريخ بالطريقة التي تناسبنا:

- **إنشاء تركيب "تاريخ":**

```
1.  struct date {
2.          int day;
3.          int month;
4.          int year;
5.  };
6.  struct students{
7.          int num;
8.          char name[20];
9.          int age;
10.         string address;
11.         double phone;
12.         date birthdate;
13. };
```

ملاحظات:

- يجب أن يكون التركيب المضمّن قبل التركيب المتضمن له، فيجب كتابة تركيب التاريخ قبل تركيب الطلاب.
- يمكن بدلا من عمل تركيب للتاريخ تعريف متغير من نوع نصي string ولكن التركيب يوفر لنا سهولة في الوصول إلى اليوم أو الشهر أو السنة .

- **إدخال قيمة لخاصية في تركيب داخلي:**

```
1.  void main() {
2.      students stud;
3.      cin >> stud.birthdate.day;
4.      cin >> stud.birthdate.month;
5.      cin >> stud.birthdate.year;
6.      cin >> stud.address;
7.      cin >> stud.phone;
8.  }
```

- برنامج لإدخال وعرض بيانات الطلاب (مصفوفة تركيب):

```
#include <iostream>
#include <string>
using namespace std;

struct date{
	int day;
	int month;
	int year;
};
struct students{
	int num;
	char name[20];
	int age;
	string address;
	double phone;
	date birthdate;
};

void main(){
	students stud[10];

	for(int i=0; i<3; i++){
		stud[i].num = i;
		cout << "Student number: " << i+1 << endl;
		cout << "---------------------\n\n";

		cout << "Name:\t\t";		cin >> stud[i].name;
		cout << "Age:\t\t";		cin >> stud[i].age;
		cout << "Address:\t";		cin >> stud[i].address;
		cout << "Phone:\t\t";		cin >> stud[i].phone;
		cout << "Birth day:\t";		cin >> stud[i].birthdate.day;
		cout << "Birth month:\t";	cin >> stud[i].birthdate.month;
		cout << "Birth year:\t";		cin >> stud[i].birthdate.year;
		system("cls");
	}
	cout << "Num\tName\tAge\tAddress\tPhone\tBirthdate\n";
	cout << "----------------------------------------\n";
	for(i=0; i<3; i++){
		cout << i+1 << "\t";
		cout << stud[i].name << "\t";
		cout << stud[i].age << "\t";
		cout << stud[i].address << "\t";
		cout << stud[i].phone << "\t";
		cout << stud[i].birthdate.day << "/" ;
		cout << stud[i].birthdate.month << "/";
		cout << stud[i].birthdate.year << "\n";
	}
}
```

ـ شاشة إدخال البيانات:

```
Student number: 1
--------------------

Name:           ali
Age:            21
Address:        sana'a
Phone:          177
birth day:      1
birth month:    1
birth year:     1989_
```

ـ شاشة عرض النتائج:

```
Num     Name    Age     Address Phone   Birthdate
-------------------------------------------------
1       ali     21      sana'a  177     1/1/1989
2       badr    22      aden    277     2/2/1988
3       omar    23      amran   377     3/3/1987
Press any key to continue_
```

واجب:
في المثال السابق أضف إمكانية للبرنامج يتيح للمستخدم أن يحدد عدد الطلاب الذين يريد أن يدخل بياناتهم.

البرمجة الموجهة (الكائنية)
Objects Oriented Programming

مقدمة:

يطلق عليها : برمجة كائنية أوغرضية الهدف أوهدفية أو شيئية..
وهي برمجة تقوم على الاهتمام بطريقة حل المشكلة قبل تحويلها الى كود..
من تقنياتها الدنيا: المصفوفات، والدوال، والتركيبات،... ومن تقنياتها العليا: المؤشرات والقوائم المتصلة.

تعريف البرمجة الكائنية:

طريقة لتجميع البيانات والوظائف (الصفات والخصائص) معاً في قالب واحد يسمى الفصيلة أو الفئة.

الصفات والخصائص Attributes & Properties:

يتكون الكائن من صفات Attributes ← متغيرات variables.
وخصائص Properties ← دوال functions.

- مكونات التركيب ومكونات الكائن:

التركيب:
- متغير
- متغير
- متغير

الكائن:
- متغير "بيان/صفة"
- متغير
- متغير
- دالة "مهمة/خاصية/طريقة"
- دالة
- دالة

أوجه الاختلاف بين التركيب والكائن:

البرمجة الكائنية تطوير للبرمجة الهيكلية فالتراكيب استمدت بناءها من الهياكل فالهيكل يتكون من جزء يتفرع منه أجزاء ترتبط به، حيث تم إضافة خصائص "طرق" للتركيب وأطلق عليه اسم كائن.

التركيب struct	الصنف class
عدم القدرة على إخفاء البيانات.	له القدرة على إخفاء البيانات.
طريقة الوصول للصفات "المتغيرات" في التركيب بشكل عام ومتاح للجميع.	لا يمكن الوصول المباشر إلى الصفات إلا باستخدام طرق معينة عبر مستويات الحماية ومن خلال الإعلان عنها.
الوصول إلى البيانات داخل التركيب يتم بواسطة متغيرات التركيب.	الوصول إلى البيانات في الصنف يتم بواسطة خصائص "دوال" الكائن.

مقارنة بين أنواع البرمجة:

البرمجة:	التقليدية	الهيكلية	الكائنية
صفاتها	- لا تعتمد أسس ولا قواعد (عشوائية) - تهتم بحل المشكلة فقط.	- تعتمد على أسس وقواعد بسيطة تتمثل في استخدام المصفوفات والدوال والتراكيب. - تهتم بتنظيم الكود البرمجي وجعله سهل التعديل.	- تعتمد أسس وقواعد لحل المشكلة. - تهتم بطريقة حل المشكلة أولاً قبل تحويلها إلى كود برمجي.
مميزاتها	- أنها تحل المشاكل الصغيرة. - تدعم عمل الفرد على البرنامج	- تحل المشاكل المتوسطة. - تدعم عمل الفرد أكثر من دعمها للفريق الواحد. - تقليل استخدام الذاكرة. - تنظم البرنامج. - جمع المتغيرات في متغير واحد (تركيب) يسهل الوصول إليه وإلى مكوناته. - تقسيم البرنامج وظيفياً إلى دوال بسيطة. - يمكن اكتشاف الخطأ بسهولة. - يمكن تطوير البرنامج بشكل متوسط.	- تحل المشاكل المعقدة. - تدعم عمل الفريق الواحد. - تنظم البرنامج . - طريقة لتقسيم البرنامج إلى برامج صغيرة وظيفية وفعالة. - سهوله القراءة والفهم والتتبع واكتشاف الأخطاء. - سهوله المعالجة والتنقيح. - حماية البيانات واستخدام طرق للوصول. - قابلية إعادة الاستعمال. - يمكن اكتشاف الخطأ بسهولة. - يمكن تطوير البرنامج بشكل كبير جداً بمجهود بسيط. - تطورت عن الهياكل struct.
عيوبها	- لا يمكن اكتشاف الخطأ في البرنامج. - لا يمكن تطوير البرنامج بسهولة. - تطوير البرنامج يزيد حجم البرنامج بشكل كبير. - لا تدعم عمل الفريق الواحد.	- يصعب تقسيم العمل لعدة مبرمجين أو مجموعات. - يمكن تطوير البرنامج إلى مدى معين ثم يصعب بعد ذلك السيطرة عليه نظراً لكثرة الدوال والتراكيب.	- تتطلب خبرة برمجية. - بناء الكائنات يحتاج إلى تحليل وتصميم وربط مع كائنات أخرى ومراعاة شروط الهيكلية والكبسلة والتجريد وغيرها..
ملاحظات	- تحل مشكلة معينة	- كل دالة تحل مشكلة والبرنامج ككل يحل مجموعة مشاكل. - التركيب يستطيع تخزين مجموعة مختلفة من المتغيرات.	- كل كائن يتولى حل مجموعة مشاكل مترابطة، ومجموعة الكائنات تتولى حل كافة المشاكل المتعلقة بالمشروع. - الكائن يستطيع تخزين مجموعة مختلفة من المتغيرات ويحتوي دوال.

الفصيلة / الصنف / الفئة (Class)

- تعريفها:
مجموعة من الأغراض أو الكيانات المتشابهة في الخصائص والسلوك.
وهي بناء برمجي يجمع "يغلف" البيانات (الصفات) ومهامها (الوظائف/الخصائص) معا.
تتركب الفصيلة من مجموعه من الصفات والخصائص.
تحتوي على البيانات مع الوظائف (الدوال) بدلا من البيانات فقط.

ملاحظة: إذا لم تخدم البيانات المهام فلا داعي لوجودها.

- حماية البيانات في الصنف:

يحتوي الصنف على سماحيات تمنع المبرمجين من الوصول إلى البيانات المحمية.
والصفات الموجودة داخل الكلاس لا يعرفها ألا من بنى الكلاس.
وللوصول إلى الصفات يتم استخدام الخصائص، كما يمكن حماية الخصائص بحيث لا يمكن الوصول إليها إلا من داخل الكائن.

الكائن:

عبارة عن تركيب لظاهرة معينة، الظاهرة :مؤسسات أوهيئات.

- تعرف الكيانات على أصنافها:

مثال:
صنف أو فصيلة الطالب:
يتكون من هذا الصنف الكثير من الكائنات مثل
الطالب/محمد والطالب/علي والطالب/احمد .....الخ

-شفرة تكوين الكائن:

1.	Class class_name	كلمة مفتاحية class واسم الفئة
2.	{	
3.	public:	منطقة الصفات والخصائص العامة
4.	data type member1;	تعريف متغير/ صفة عامة
5.	data type function_name(per)	تعريف دالة / خاصية عامة
6.	…. ….. …..	
7.	Private:	منطقة الصفات والخصائص المحمية
8.	data type member1;	تعريف متغير/ صفة محمية
9.	data type function_name(per)	تعريف دالة / خاصية محمية
10.	…. ….. …..	
11.	Protected:	منطقة الصفات والخصائص المورثة
12.	data type member1;	تعريف متغير/ صفة يمكن توريثها
13.	data type function_name(per)	تعريف دالة / خاصية يمكن توريثها
14.	…. ….. …..	
15.	};	

- المستوى العام public

كلمة مفتاحية تسمح للوصول لكل عنصر في الصنف ولكل المستخدمين.

- المستوى الخاص private

كلمة مفتاحية لا تسمح للوصول لعناصر الصنف إلا عن طريق الصنف نفسه (عن طريق الدالة المعرفة في الصنف).

- المستوى المحمي protected

كلمة مفتاحية تسمح لاشتقاق عناصر وتوريثها لأصناف أخرى، حيث أنها لا تسمح للوصول لعناصر الصنف إلا عن طريق الصنف نفسه مثل المستوى private أو عن طريق الصنف المشتق (دالة في الصنف المشتق).

ملاحظات:

- يفضل أن يكتب اسم الصنف بحرف كبير (يبدأ بحرف كبير) لتمييزه عن التركيب.
- تكتب الفئة قبل الدالة الرئيسية.
- يتم تعريف كائن على "الفئة/الصنف" داخل الدالة الرئيسية.
- المتغيرات (الصفات) والدوال (الخصائص) المصرح في الجزء private لا يمكن استخدامها خارج الصنف.
- كل ما يندرج تحت المستوى public يمكن استخدامه داخل الصنف وخارجه.
- في حاله عدم ذكر public أو private فان الكلمة الافتراضية هي private.

حجم الكائن:

كلاً من الكائن والتركيب يحجزان مساحة مركبة في الذاكرة حجمها هو حجم جميع المتغيرات الداخلة في الكائن أو التركيب.

مثال :

التركيب : لنأخذ مثال الطلاب الموجود في الفصل السابق:

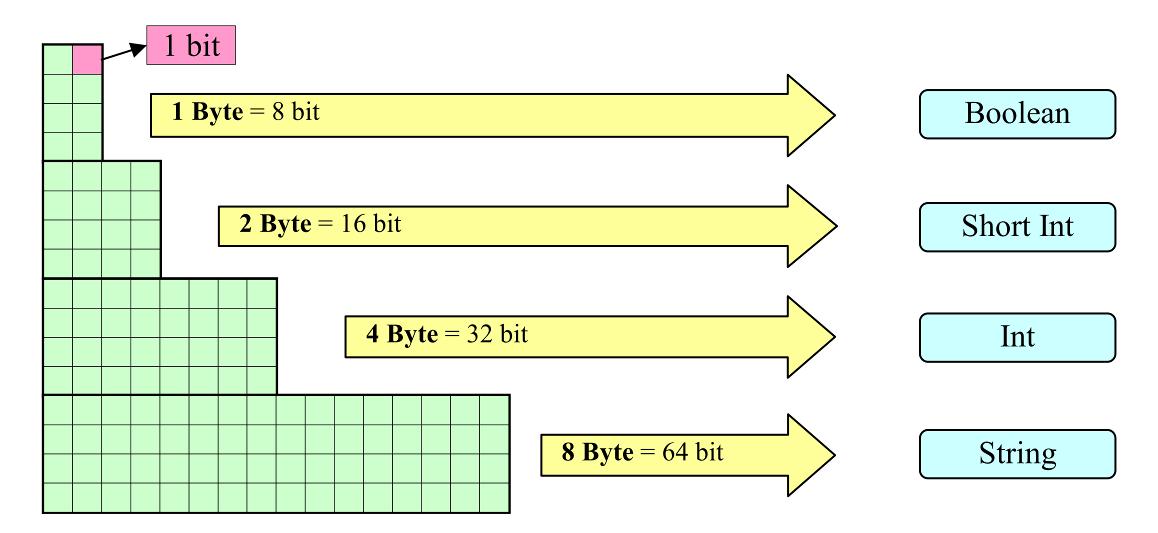
```
struct students{
        int num;
        char name[20];
        int age;
        string address;
        double phone;
};
```

Int	Char	Int	String	Double	الإجمالي
4	20	4	8	8	**44 byte**

الكائن: يشبه التركيب بالإضافة إلى أنه يحجز مساحات للمتغيرات التي يتم تعريفها داخل الصفات "الدوال" عند استخدامها.

أمثلة من الحياة الواقعية:

توزيع المساحات على المتغيرات:



المتغيرات:
في الواقع إذا اعتبرنا أن الرام قطعة أرض وأنها مقسمة إلى لبنات، فإنه لبناء منزل صغير نحتاج إلى ٨ لبنات ولبناء منزل كبير (بحجم منزلين صغيرين) فإنك بحاجة إلى ١٦ لبنة ولبناء شركة صغيرة (بحجم ٤ منازل صغيرة) تحتاج إلى ٣٢ لبنة ولبناء شركة متوسطة أو ملعب (بحجم ٨ منازل صغيرة) فإننا نحتاج إلى ٦٤ لبنة.

فإنه في الكمبيوتر يمكننا تشبيه المنزل الصغير بـ (Boolean) والمنزل الكبير بـ (Short Int) والشركة الصغيرة بـ (Int) والشركة المتوسطة بـ (String) والملعب بـ (Double).

التركيب:
بينما التركيب مثل المؤسسة يحتوي مباني مختلفة الأحجام كل مبنى يحجز عدد معين من اللبنات فالبوابة تحتاج إلى ٨ لبنات والمخازن تحتاج إلى ١٦ لبنة والمكاتب تحتاج إلى ٣٢ لبنة وكل هذه اللبنات موجودة في مكان واحد "متجاورة".

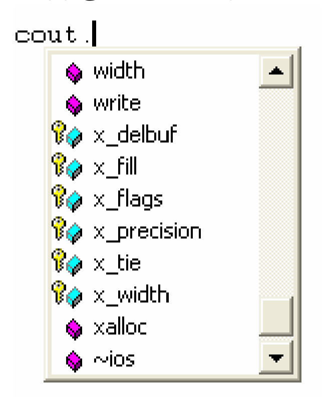
الكائن:
عبارة عن تركيب لكن يحتوي على وظائف، وعند إنشاء كائن من نوع صنف معين فإن الأمر يشبه المثال التالي:
(عندما تشتري كتاب طبخ فإن الوصفة تمثل "الصنف أو الكلاس" بينما الطبق الذي تنتجه هو الكائن فأنت لا تستطيع أكل الوصفة بل الطعام "الكائن").
لذلك فإننا نقول أن هذه الطبخة من تلك الوصفة وهذا الكائن من ذاك الصنف.

ملاحظات:
يطلق على متغيرات الصنف أسماء (خصائص، صفات، بيانات)
يطلق على دوال الصنف أسماء (طرق، مناهج، وظائف، تصرفات، مهام)

مثال عملي:

أثناء تعاملنا مع بيئة C++ استخدمنا خصائص الكائن cout لتنفيذ أعمال مختلفة وكنا نطلق عليه مجازاً دالة لكنه كائن يحتوي عدة وظائف "دوال" وهو كائن من صنف الإخراج ostream.

تظهر الصفات والخصائص في قائمة منسدلة بمجرد الضغط على زر النقطة ".":



إذاً الوظيفة width تأخذ ممرر "وسيط" واحد وتقوم بعمل مسافات بعدد الرقم الممرر.

```
cout.width(10);  // print 10 spaces
```

كذلك الكائن cin وهو من صنف الإدخال iostream.
لاحظ تقسيم الإعمال على الكائنات، كائن يعالج مسائل الإدخال وآخر يعالج مسائل الإخراج، وهذا ما يميز البرمجة الكائنية.

ملاحظات:

تأخذ القائمة المنسدلة عدة ألوان تعبر عن عمل كل عنصر فيها:

- الصفة (المتغير) تظهر بلون سماوي.
- الخاصية (الدالة) تظهر بلون وردي.
- وإذا كانت الصفة أو الخاصية محمية فإنه يظهر بجوارها رمز القفل.
- أما إذا كانت الصفة أو الخاصية مورثة فإنه يظهر بجوارها رمز المفتاح.
- الصفة العامة يمكن تغيير قيمتها بوضع علامة (=) مثال:

```
obj.var = value
```

- الخاصية العامة يمكن تغيير قيمتها عن طريق تمرير وسطاء للدالة مثال:

```
obj.function(value1, value2)
```

مثال بسيط:

```
1.  Class Simple_math
2.  {
3.  private:
4.          int r;
5.  
6.          double result()
7.          {   return r;   };
8.  public:
9.          void sum(int a, int b)
10.         {   r = a + b;   };
11. 
12.         void divided(int a, int b)
13.         {    if (b==0){
14.                  cout << "Can't divided by ziro";
15.                  r = 0;
16.              } else {
17.                 r = a / b;
18.              }
19.         };
20.         void print ()
21.         {   cout << "\n====\n"
22.                   << result
23.                   << "\n====\n" ;
24.         };
25. };
26. 
27. void main(){
28.         Simple_math  sm;
29.         sm.sum( 3 , 2 );
30.         sm.print();                                          الناتج : 5
31.         sm. divided ( 3 , 0 );                               الناتج :
32.         sm.print();                          Can't divided by ziro
33. }
```

نلاحظ أن الكلاس Simple_math يحوي إجراءين الأول لجمع عددين والإجراء divided لقسمة عددين وهو يقوم أيضاً بمعالجة مشكلة القسمة على صفر ويعيد رسالة تفيد بعدم إمكانية القسمة على صفر..
ودالة خاصة "محمية" result تقوم بإرجاع قيمة الخاصية r ، والإجراء print والذي يستدعي الدالة result ويطبعها بشكل منسق.

ملاحظات:

يفضل استخدام حرف كبير في بداية اسم الكلاس لتمييزه عن التراكيب.
يمكنك إضافة خصائص أخرى لهذا الكلاس مثل الطرح والضرب.
يمكن نقل الدالة result من القسم الخاص إلى القسم العام وبالتالي تستطيع استخدامها لاسترجاع قيمة r.

مثال لصنف الموظفين:

```
#include <iostream>
#include <string>
using namespace std;

class Employees
{
private:
        int no;
        string name;
        string address;
        int phone;
        string department;
        int salery;
        string date;
///////////////////////
public:
        void insert()
        {
                cout<< "no:";           cin >> no;
                cout<< "name:";         cin >> name;
                cout<< "address:";      cin >> address;
                cout<< "phone:";        cin >> phone;
                cout<< "department:"; cin >> department;
                cout<< "salery:";       cin >> salery;
                cout<< "date:";         cin >> date;
        };
        void print()
        {
                cout    << "n= "            << no               << "\n "
                        << "name = "        << name             << "\n "
                        << "address = " << address          << "\n "
                        << "phone= "        << phone            << " "
                        << "department = " << department    << "\n "
                        << "salery = "      << salery           << "\n "
                        << "date = "        << date             << endl;
        };
};
main(){
        Employees emp[200];
        int sum=0;

        cout << "How records you want to enter:";
        cin >> sum;
        for(int i=0; i<sum; i++){
                emp[i].insert();
        };
        cout << "==============\n";
        for( i=0; i<sum; i++){    emp[i].print();    };
}
```

مبادئ البرمجة الهيكلية:

- تحليل المشكلة (جمع المعلومات عنها).
- تصميم الهيكل.
- برمجة الحل.
- تجربة البرنامج.
- التوثيق.

ملاحظة: سيتم دراسة مبادئ البرمجة الهيكلية بشكل أكبر في الفصل الثاني.

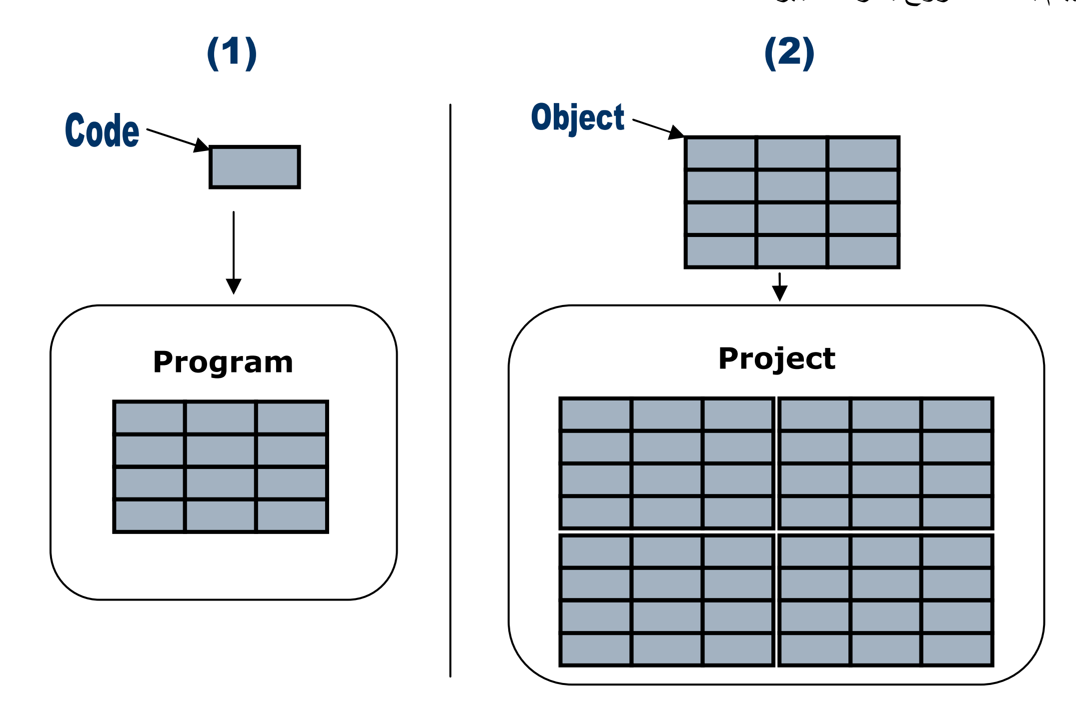
أوجه الاختلاف بين البرمجة بالدوال والبرمجة بالكائنات:

البرمجة بالدوال	البرمجة بالكائنات
يمكن بواسطتها بناء برنامج متكامل.	يمكن بواسطتها بناء مشروع متكامل.
تقسم البرنامج إلى أجزاء.	تقسم المشروع إلى برامج مستقلة فرعية.
البرنامج ضمن ملف واحد يحوي كل الدوال	كل برنامج مستقل خارج الملف الرئيسي، ويتم استدعاء البرنامج عند الحاجة إليه.
تحتاج من واحد إلى ثلاثة مبرمجين لبناء البرنامج	تحتاج إلى مبرمج أو أكثر لكل برنامج مستقل
يشترك المبرمجين في بناء البرنامج الواحد	كل مبرمج أو فريق مبرمجين مسئول عن برنامجه.
إذا وجد خطأ في دالة يتوقف البرنامج بأكمله	إذا وجد خطأ في برنامج فرعي، يستمر البرنامج الرئيسي للمشروع في العمل.
يحتوي البرنامج على عدة دوال	قد يحتوي البرنامج على أكثر من كائن.
تعمل الدوال بشكل مستقل عن بعضها	تتكامل الدوال "الخصائص" في انجاز المهمة الواحدة. وتتكامل الكائنات بواسطة الوراثة أو تعدد الواجهات.
الكود يشكل جزء من البرنامج	الكائن يمثل جزء من المشروع

ملاحظة: إذا تم وضع الكلاسات داخل الملف الأساسي ولم توضع في برامج فرعية فإن الكثير من مميزات البرمجة بالكائنات ستختفي!

كتل البرنامج الهيكلي:

يتم بناء البرنامج كود بعد كود لتكوين الكائن (ككتلة واحدة) وبعدها يتم تكوين المشروع كتلة بعد كتلة وبالتالي يتم بناء المشروع بسرعة كبيرة.



أمثلة :

- في نظام تشغيل windows :
"برنامج المفكرة" جزء من نظام ويندوز لكنه برنامج مستقل، فإن حصل عطل في المفكرة فلن يتوقف نظام التشغيل.كذلك برنامج الرسام والحاسبة... وغيرها.
- "برامج office" نلاحظ أنه يتم تعديل تصميمها بشكل كامل مع بقاء كل الإمكانيات وإضافة إمكانيات جديدة لها في كل إصدار جديد من إصدارات office وبشكل سريع وذلك نظراً لسهولة التطوير باستخدام الكائنات.

تقنيات البرمجة الموجهة

إخفاء البيانات Data Hiding
الكبسلة والتضمين Encapsulation
توريث الصفات Inheritance
تجريد البيانات Abstracter
تعدد الاستخدام Polymorphism

المشيدات والمدمرات (Constructor & Destructors)

المشيدات :

ـ تعريفها:
هي عبارة عن دالة لها نفس اسم الصنف تقوم بتزويد الكائن أو عناصره بقيم ابتدائية "تهيئة الكائن للاستخدام".

ـ الخواص:
١. لها نفس اسم الصنف.
٢. لا يمكن استدعائها من خارج الصنف.
٣. يتم تنفيذها عند إنشاء الكائن.
٤. يمكن عمل ممرات "تمرير وسطاء" أو عدم تمريرها لكنها لا تعود بشيء.
٥. يمكن إنشاء أكثر من دالة مشيد في الصنف الواحد "Overload" تحميل زائد.

ملاحظات:
المشيد يهيئ الصنف للاستخدام الأولي بإعطائه قيم أولية.
بدون المشيد لا يمكن تهيئة الصنف لأنه لم يحجز مساحة أصلاً.

مثال:
في المثال السابق الخاص بالكلاس Simple_math ماذا سيحدث لو أننا استخدمنا الخاصية print دونما الخصائص الأخرى :

```
Simple_math sm;
sm.print();
```

الناتج:

```
====
-8.58993e+008
====
```

نلاحظ أنها ستطبع قيمة عشوائية "-8.58993e+008" وذلك لأنه يطبع قيمة المتغير r بدون أن يسند له أي قيمة.
لذلك فنحن بحاجة لطريقة لإسناد قيمة ابتدائية للمتغير r.
تقبل بعض لغات البرمجة إسناد قيمة للمتغير (الصفة) مباشرةً لكن بيئة C++ تشترط أن القيمة الابتدائية يتم إسنادها من خلال دالة وإلا ستظهر لك الرسالة التالية:

'r' : pure specifier can only be specified for functions

'r' : pure specifier can only be specified for functions

مثال عملي:
عند تعاملك مع بيئة visual basic فإنك عندما تنشئ نموذج "Form" فإنه يحتوي على خصائص مسبقاً دون تدخل منك ومنها "height" و "width" وبدون وجود القيم الابتدائية لهذا النموذج فلن تستطيع مشاهدة النموذج لان ارتفاع النموذج وعرضه سيكونان "صفراً".

المدمرات / الهادمات :

- تعريفها:

عبارة عن دالة تقوم بإزالة الأغراض والكيانات من الذاكرة (إلغاء الكائن من الذاكرة).

- الخواص:

١. لها نفس اسم الصنف.
٢. يسبق اسم دالة الهدم العلامة (~).
٣. لا توجد لها ممررات ولا تعود بشيء
٤. تقوم بإلغاء المساحات المحجوزة للكائنات في الذاكرة.
٥. لا يمكن إنشاء أكثر من دالة هدم واحدة.
٦. يتم تنفيذها عند إلغاء الكائن أو انتهاء البرنامج.

شفرة المشيد:

```
1.  Class Simple_math
2.  {
3.  private:
4.        int r;
5.
6.        double result()
7.        {   return r;   };
8.  public:
9.        Simple_math()                    دالة المشيد ١
10.       {   r = 0;   };
11.
12.       Simple_math(int x)               دالة المشيد ٢ (تحميل زائد)
13.       {   r = x;   };
14.
15.       void sum(int a, int b)
16.       {   r = a + b;   };
17.       void print ()
18.       {   cout << "\n====\n"
19.              << result
20.              << "\n====\n" ;
21.       };
22. };
23.
24. void main(){
25.       Simple_math  sm;                 إنشاء كائن sm
26.       sm.print();                      الناتج : 0
27.       Simple_math sm2(1);              إنشاء كائن sm2 وإسناد قيمة ابتدائية له
28.       sm2. print();                    الناتج : 1
29. }
```

تم حذف دالة divided لاختصار البرنامج.

إضافة شفرة المدمر:

```
1.  Class Simple_math
2.  {
3.  private:
4.          int r;
5.  
6.          double result()
7.          {   return r;   };
8.  public:
9.          Simple_math()
10.         {   r = 0;   };
11. 
12.         Simple_math(int x)
13.         {   r = x;   };
14. 
15.         void sum(int a, int b)
16.         {   r = a + b;   };
17.         void print ()
18.         {   cout << "\n====\n"
19.                 << result
20.                 << "\n====\n" ;
21.         };
22.         ~Simple_math()                              دالة المدمر
23.         {
24.            cout << "\n*****"
25.                 << " The End *****\n";
26.         };
27. };
28. 
29. void main(){
30.         Simple_math  sm;                            إنشاء كائن sm
31.                             عند  إغلاق نافذة البرنامج ستظهر العبارة :
32.                                          ***** The End *****
33. }
```

ملاحظات:

- يتم التفريق بين دالة الهدم عن دالة المشيد بعلامة ~ تسبقها ، ولا يمكن استدعائها فهي تنفذ نفسها عند إنهاء البرنامج أو حذف الكائن.
- عند إغلاق البرنامج يجب حذف جميع المتغيرات التي يستخدمها من الذاكرة، وإلا فإنها ستبقى حتى تملأها وبالتالي سنحتاج إلى إضافة ذاكرة إضافية والتي بدورها ستمتلئ، يمكنك ملاحظة أنه عند إعطاء المتغير x قيمة معينة ثم إعادة تشغيل البرنامج وطباعة x فإنها لا تعيد شيئاً..
- تقوم بيئة C++ بتدمير الكائن مباشرة عند انتهاء البرنامج ولكن إذا أردنا التحكم بحذف الكائن فإننا ننشئ دالة هدم خاصة بنا.
- المدمرات مهمة جداً وتوجد في كل لغة برمجية دوال مدمرة، يمكنك ملاحظة دالة التدمير الخاصة ببيئة C++ عند ظهور رسالة "Press any key to continue" في نهاية تنفيذ البرنامج.
- تقوم دالة الهدم الخاصة بنا بتدمير الكائن قبل دالة الهدم الخاصة بالبيئة.

مثال لصنف الطالب:

```
#include<iostream>
#include<string>
using namespace std;
/////////////////////////////////////////////
class Student
{
private:
        int     num,   level;
        string  name,  space;
//////////////////////////////////////////////
public:
        void assigned(int n, string m, int l, string s)
        {
                num=n ; level=l ;
                name=m ; space=s ;
        };
        void print( )
        {       cout    <<num<<ends
                        <<name<<ends
                         <<level<<ends
                        <<space<<endl;
        };
//////////////////////////////////////////////////
        Student( )
        {       num=0; name="nodata"; level=0; space="nodata";   };

        Student(int n, int l, string m, string s)
        {       num=n; level=l; name=m; space=s;   };
//////////////////////////////////////////////////
        ~ Student( )
        {cout<<"the end\n";};
};
main()
{
        Student stud;
        Student studl(1,4,"ali","it");
        stud.print();
        studl.print();
        int a,b;
        string c,d;

        cout << "Enter Number: ";     cin >> a;
        cout << "Enter Level: ";      cin >> b;
        cout << "Enter name: ";       cin >> c;
        cout << "Enter space: ";      cin >> d;
        stud.assigned( a, c, b, d);
        stud.print();
}
```

إنشاء دوال الصنف خارجه :

```
1.	#include<iostream>
2.	#include<string>
3.	using namespace std;
4.	//////////////////////////////////////////////
5.	class Student
6.	{
7.	private:
8.		int	num,	level;
9.		string  name,  space;
10.	//////////////////////////////////////////////
11.	public:
12.		void assigned(int, string, int, string);
13.
14.		void print()
15.		{	cout	<<num<<ends
16.				<<name<<ends
17.				 <<level<<ends
18.				<<space<<endl;
19.		};
20.	//////////////////////////////////////////////////
21.		Student( ){	num=0; name="nodata"; level=0; space="nodata";	};
22.
23.		Student(int n, int l, string m, string s){
			num=n; level=l; name=m; space=s;
		};
24.	//////////////////////////////////////////////////
25.		~ Student( )
26.		{cout<<"the end\n";};
27.	};
28.
29.	void Student::assigned(int w , string x, int y, string z) {
30.		num = w; name = x; level = y; space=z;
31.	};
32.
33.	void main( ){
34.		Student stud;
35.		Student studl(1,4,"ali","it");
36.		stud.print();
37.		stud1.print();
38.	int a,b;
39.	string c,d;
40.
41.	cout << "Enter Number: ";	cin >> a;
42.	cout << "Enter Level: ";	cin >> b;
43.	cout << "Enter name: ";	cin >> c;
44.	cout << "Enter space: ";	cin >> d;
45.		stud.assigned(a,c,b,d);
46.		stud.print();
47.	}
```

* لاحظ السطر رقم ١٢ :

١- يمكن أن نحذف أسماء المتغيرات ونبقي أنواعها فقط:

assigned(int, string, int, string);

٢- ويمكن أن يحتوي أسماء المتغيرات:

void assigned(int n, string m, int l, string s)

مثال:

class simple_math{ public: int getSum(int a, int b){ return a+b; }; }; ///////////////////////////////// main(){ simple_math sm; cout << sm.getSum(1,2) int getSum (int a, int b)	class simple_math{ public: int getSum(int ,int); }; ///////////////////////////////// main(){ simple_math::getSum(int a, int b){ return a+b; }; simple_math sm; cout << sm.getSum(1,2) int getSum (int ,int)

توفر الطريقة في اليسار حماية أكبر للبيانات كما في المثالين السابقين لاحظ أنه لا يتم عرض أسماء المتغيرات.

* لاحظ السطور من ٢٩ إلى ٣١:

١- يمكننا إعادة تسمية المتغيرات بأسماء تختلف عن الأسماء الموجودة في الدالة داخل الكلاس..

void Student::assigned(int w , string x, int y, string z) { }

٢- يمكن الوصول إلى متغيرات الكلاس داخل هذه الدالة الخارجية:

num = w; name = x; level = y; space=z;

مميزات إنشاء دوال الصنف خارجه:

- إضافة تعدد أشكال للدوال الموجودة داخل الكلاس.
- تطوير البرنامج مثل "حزم برامج office التطويرية".

ملاحظة: يمكن إنشاء دالة خارج الصنف أسفل الدالة الرئيسية main

```
#include <iostream.h>

class simple_math{
public:
        int getSum(int ,int );
        double pow3(int);
};
////////////////////////////////
int main(){

    simple_math sm;

    cout << sm.getSum(1,2);
    cout << sm.pow3(5);
};
////////////////////////////////

int simple_math::getSum( int a,  int b){
      return a+b;
};

#include <math.h>
double simple_math::pow3( int x){
    return pow(x , 3);
};
```

إنشاء المكتبات
Build Libraries

تحتوي المكتبة على فئات وتراكيب ولا تحتوي على أوامر تنفيذية مثل الطباعة على الشاشة وبالتالي لا يمكن تنفيذ المكتبة ولكن يمكن تضمينها "استيرادها" في ملف "source file" ثم تعريف بيانات من نوع الفئات الموجودة في المكتبة.

يتم تسمية أي مكتبة بأحرف كبيرة وتأخذ الامتداد ".h" مثال : MYLIBRARY.h.

يمكن وضع مكتبتنا داخل مجلد "include" الخاص ببيئة C++ في المسار التالي:
C:\program files\microsoft visual studio\vc98\include

وسيكون تضمين المكتبة بين قوسين #include <mylibrary.h>

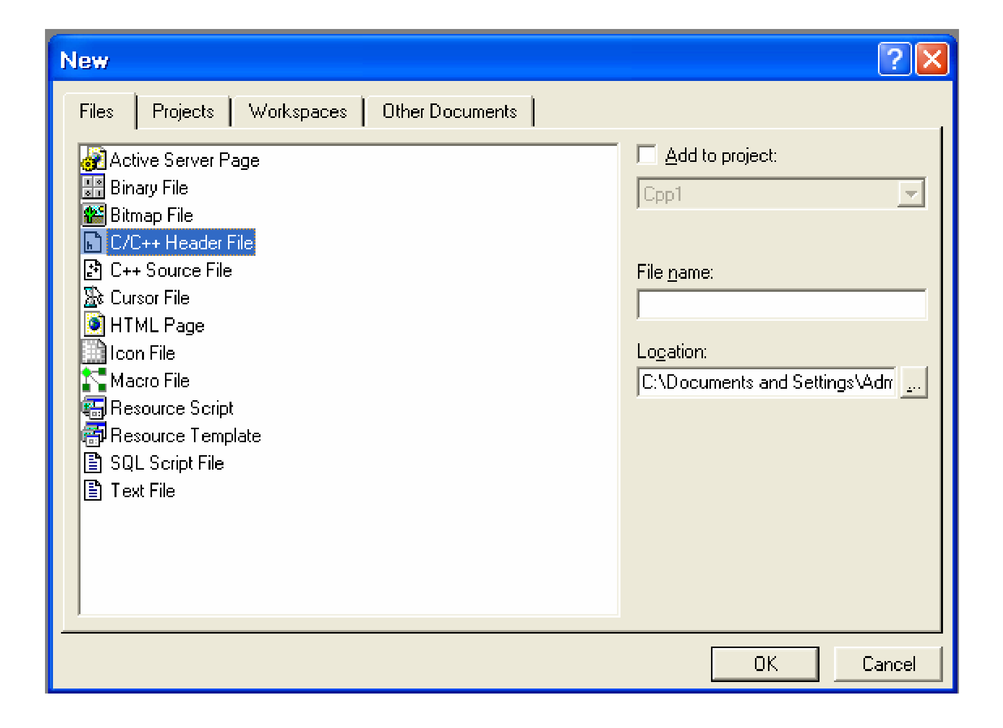
كما يمكن وضع مكتبتنا جوار ملف "source file" الخاص بنا وعندها نستخدم إشارتي تنصيص #include "mylibrary.h".

القوسين <mylibrary.h> تعني أن تبحث بيئة C++ عن المكتبة في مجلد include.
عند تحديث المكتبات فإن الامتداد h. يبقى في المكتبات التي تنشئها.

- لاحظ أنه يجب إرفاق مكتبتك مع البرنامج عند نقله إلى إي جهاز آخر للعمل.

خطوات عمل مكتبة :

١- إنشاء ملف مكتبة جديد:



٢- قص الكلاس من ملف "source file" ولصقه في ملف المكتبة (مع إضافة المكتبات والتحديثات اللازمة لعمل مكتبتك):

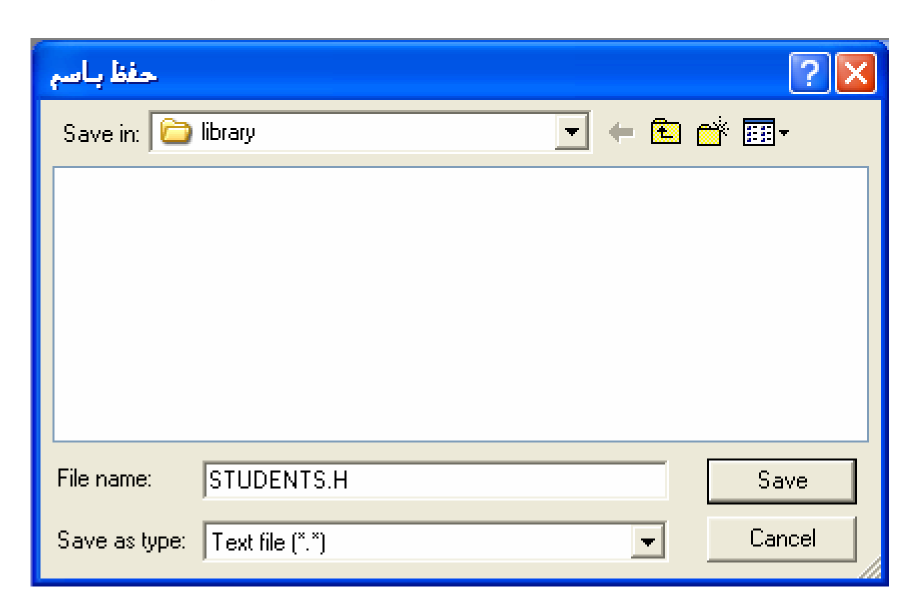
```
#include<iostream>
#include<string>
using namespace std;
////////////////////////////////////////////////
class Student
{
private:
    int     num,   level;
    string  name,  space;
//--------------------------------------------//
public:
    void assigned(int, string, int, string);

    void print()
    {   cout    <<num  <<ends
                <<name <<ends
                <<level<<ends
                <<space<<endl;
    };
//--------------------------------------------//
    Student( ){ num=0; name="NoData"; level=0; space="NoData";  };

    Student(int n, int l, string m, string s){
        num=n; level=l; name=m; space=s;
    };
//--------------------------------------------//
    ~Student( )
    { cout<<"the end\n"; };
};
```

(ملف المكتبة STUDENTS.H)

٣- حفظ ملف المكتبة بأحرف كبيرة وبامتداد h. جوار ملف "source file" أو في مسار "include".



تم حفظ الملف داخل مجلد library جوار ملف source file (prog1.cpp).



شكل ملف المكتبة:



٤- تضمين المكتبة في ملف "source file" ويمكنك إنشاء دوال للصنف من خارجه:

```
#include<iostream>
#include<string>
#include "library/students.h"
using namespace std;

/////////////////////////////////////////////////////////////////
void Student::assigned(int w , string x, int y, string z) {
    num = w; name = x; level = y; space=z;
};

/////////////////////////////////////////////////////////////////

void main( ){
    Student stud;
    Student stud1(1,4,"ali","it");
    stud.print();
    stud1.print();

    int a,c;
    string b,d;

    cout << "Enter Number: ";   cin >> a;
    cout << "Enter name: ";     cin >> b;
    cout << "Enter Level: ";    cin >> c;
    cout << "Enter space: ";    cin >> d;

    stud.assigned(a,b,c,d);
    stud.print();
}
```

(ملف البرنامج source file [prog1.cpp])

ملاحظات:

- يجب تضمين المكتبات مرة أخرى في ملف "source file".
- يمكن إنشاء دوال للصنف من خارجه لاحظ :

```
void Student::assigned(int w , string x, int y, string z) {
        num = w; name = x; level = y; space=z;
};
```

بشرط أن تكون الدالة مصرح عنها في الصنف:

```
void assigned(int, string, int, string);
```

الفصل
13

الوراثـــة
The Inheritance

مقدمة :
تعد الوراثة واحدة من أهم الخصائص في البرمجة الكائنية، لأنها تعطيك مرونة وقوة في كتابة برامجك، كما أنها تسهل عملية تطوير البرنامج، وتجعل من كتابة الأكواد متعة بالفعل، لكنها تحتاج بالمقابل إلى بذل مجهود فكري لبناء الصنف[1] ومعرفة كيف سيتم توريث الأصناف المشتقة، وكذلك معرفة بعض المفاهيم مثل التجريد وتعدد الأشكال، ومتى ما تعمقت معرفتك بالبرمجة الكائنية فسيكون لديك سلاح قوي لا يمكنك أن تتخلى عنه.

التعريف:
- عبارة عن توريث صفات وعناصر من صف المورث إلى صف الوريث.
- نقل عناصر صنف إلى صنف آخر.
- تعريف صنف بدلالة صنف آخر.

يعتبر صنف المورث (صنف الأب) الصنف الأساس Base class
بينما يعتبر صنف الوريث (صنف الابن) الصنف المشتق Derived class

شفرة الكائن المشتق:

```
1.  Class Derived_class : Access Base_class
2.  {
3.  private:
4.          Declarations…
5.  Public:
6.          Declarations…
7.  Protected:
8.          Declarations…
9.  }
```

شرح الشفرة:

Class	إنشاء كلاس جديد
Drived_class	اسم الكلاس الإبن
Access	طرق الوصول إلى صنف الأساس "الأب" وتكون على إحدى حالتين: :Public نقل الصفات كما هي. :Private نقل الصفات وتحويل الصفات العامة والمورثة إلى محمية.
Base_class	اسم الكلاس الأب ، ويجب أن يكون الكلاس الأب موجوداً قبل إنشاء الكلاس الإبن.

(1) راجع الفصل ١٥ في كتاب "تطبيق UML التحليل والتصميم بالمنحى للكائن باستخدام UML " ترجمة وإعداد "خالد الشقراوي".

ملاحظات:

- الصفات المحمية في الأب تبقى محمية في الابن وتبقى غير متاحة في الابن حتى مع تحديد الوصول "Access" إلى عام Public.
- الصفات الموروثة Protected متاحة داخل الصنف الابن ومحمية خارج الصنف الأب والابن.
- يمكن أن يورث الصنف الأساس عدد من الأصناف المشتقة، ويمكن للصنف المشتق أن يرث من عدة أصناف أساس.
- عند عدم الرغبة في توريث صفة يحملها الصنف الأساس إلى الأصناف المشتقة فإننا نجعلها محمية Private.
- المستوى public يسمح باستخدام الصفات والخصائص داخل الأصناف وخارجها وداخل الأصناف المشتقة وخارجها (أي أنها تكون متاحة).

أمثلة من الواقع:

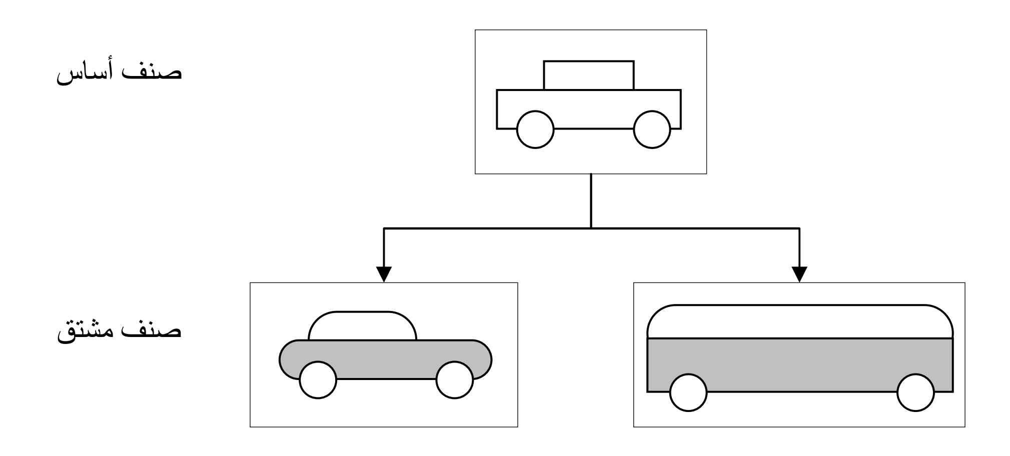
مثال ١:

شركة السيارات تقوم بعمل قالب واحد لمجموعة من السيارات وفي المرحلة التالية يتم إضافة بعض التفاصيل إلى السيارات لتمييزها مثل اللون ونوع التنجيد وقوة المحرك... الخ
نستطيع أن نقول أن القالب العام هو الأب وأن السيارات المختلفة ورثت كل الصفات الأساسية من الصنف الأب وأضيف لكل واحدة منها صفات مختلفة.

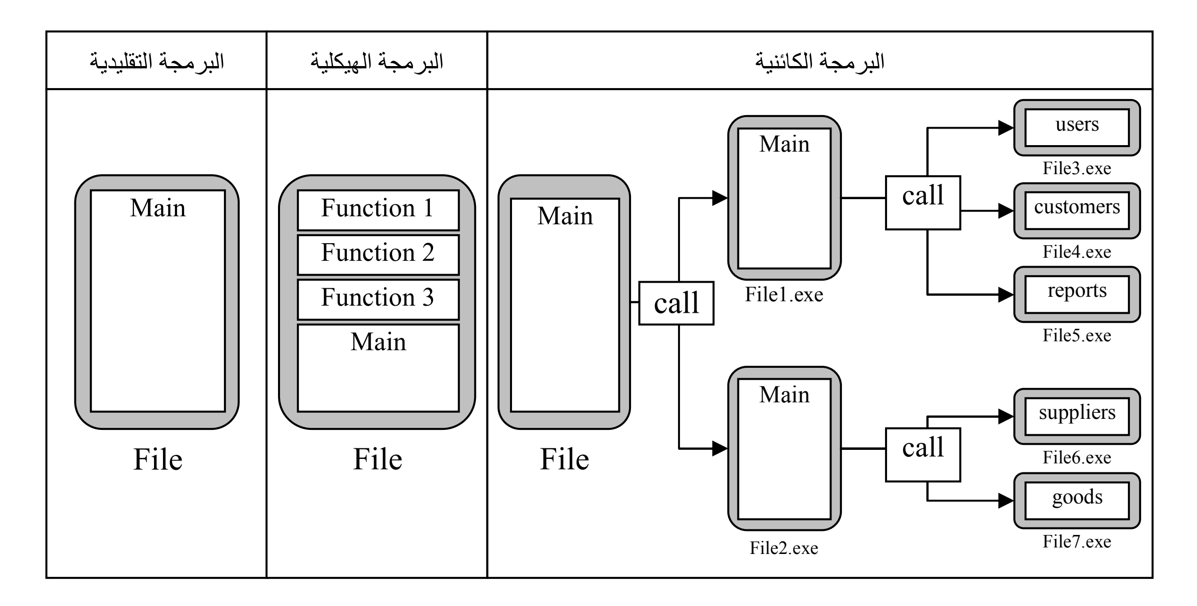
مثال ٢:



في بعض الألعاب الالكترونية هناك العديد من السيارات قد تصل إلى أكثر من ٥٠ نوع فكيف يتم ذلك؟ هل يتم عمل كلاس لكل سيارة على حدة؟ فمن الصعب عمل ذلك لأن لكل سيارة شكل ولون وحجم وسرعة تختلف من واحدة لأخرى.. من ناحية أخرى فإن كل السيارات لها ٤ عجلات ولها نفس الحركة "يمين ويسار أمام خلف" ونفس دوال التسارع ودوال التحطم ودوال تشغيل الإضاءة والمكابح... الخ.
ومن هنا يأتي دور الوراثة ليحل هذه الإشكالية.. حيث يتم عمل صنف أب يحتوي كل الخصائص والصفات الأساسية التي تشترك فيها كل السيارات، ثم يتم برمجة أصناف مشتقة تحتوي على إضافات وتغييرات بسيطة في الشكل والحجم والتسارع وقوة التحمل مثلاً بينما ترث كل الصفات الأساسية من الصنف الأب.



عودة إلى أنواع البرمجة:



البرمجة الهيكلية:

تعتمد على إحدى طريقتين:
١) طريقة البرمجة المتتالية: حيث أن المبرمج يقوم ببرمجة بما يخصه ثم ينتقل البرنامج إلى مبرمج آخر ليقوم بدوره بما يخصه وهكذا دواليك..

٢) طريقة البرمجة المستقلة: حيث يعمل كل مبرمج على جزء من البرنامج ثم يتم تجميع الأجزاء في برنامج واحد.

من عيوبها:
- التأخير في تنفيذ البرنامج.
- كل مبرمج يجب أن يفهم ما عمله زميله.
- أي خطأ في دالة يوقف البرنامج.
- أي تعديل في برنامج فرعي يحتاج لتعديل في البرنامج الرئيسي.

البرمجة الكائنية:

أجزاء البرنامج عبارة عن برامج فرعية مستقلة.
كل برنامج فرعي يعمله مبرمج أو فريق.
تطوير البرنامج الفرعي لا يعني هدمه وبناء برنامج جديد، ولكن يعني إضافة وظائف أخرى.
إذا كبر حجم البرنامج الفرعي مع كثرة التطويرات فيه فيمكن تقسيمه إلى برامج فرعية عن طريق التوريث وإضافة وظائف "دوال" أخرى.
يجب عدم تداخل الوظائف فكل وظيفة يجب أن تختص بعمل شيء معين.
إضافة وظائف للبرنامج لا تؤثر على الوظائف الرئيسية.

مميزاتها:
- كل المبرمجين يعملون في نفس الوقت.
- يمكن لكل مبرمج أن يبرمج بأي لغة.
- كل برنامج له حماية لبياناته.
- التعديل في برنامج فرعي لا يحتاج إلى التعديل في البرنامج الرئيسي.
- إذا تعطل برنامج فرعي فلا يتوقف البرنامج الرئيسي.

مثال : لكائن سيارة يحسب الأمتار التي تقطعها ويظهر المسافة المقطوعة بالكيلو متر..

```
#include <iostream>
#include <string>
using namespace std;

class movement {
	//////////////////////////////
	private :
		int   _state;    float _kelo;
		float _meter;    int   _position;  int _side;
	//////////////////////////////
	public :
		movement(){
			_state = 0; _kelo = 0;
			_meter = 0; _position = 0;   _side = 1;
		}
		//-------------------//
		void power_on(){
		   _state = 1;
		   cout << "START\a\n";
		}
		//-------------------//
		void move_on( float meter, char side){
			if (_state == 0 ){
				cout << "You must start the car!\n";
			}else{
				cout << "\nMOVE: ";
				if(side=='f'){
					_meter += meter;
					_side = 1;

					if(_meter>= 100) {
						_meter -= 100;
						_kelo++;
					}
					cout << "\t-> Moving " << meter
						<< " meter forward\n";
				}else if(side=='b'){
					_meter -= meter;
					_side = -1;

					if(_meter<= 0){
						_meter=99;
						_kelo--;
					}
					cout << "\t-> Moving " << meter
						<< " meter backward\n";
				}
			}
		}
		//-------------------//
		void position_(){
			if (_side==1){
				cout << "\t   You moved to " << _kelo << ","
					<< _meter << " km\n";
			}else{
				cout << "\t   You back to " << _kelo << ","
					<< _meter << " km\n";
			}

		}
```

```
            ~movement(){
                cout << "\nSTOP\n";
            }
        ////////////////////////////
};

//############################################//
class shape : public movement {
        ////////////////////////////
        private :
                string _model;      string _color;
                int _doors;         int _engine_force;
        ////////////////////////////
        public :
                shape(){
                 _model = "BMW";   _color = "white";
                 _doors = 4;        _engine_force= 200;
                }
                 //------------------//
                void recolor( string color ){
                    _color = color;
                }
                 //------------------//
                void upgread_engine( int force){
                    _engine_force += force;
                }
                 //------------------//
                void descripe_(){
                    cout << "Model:\t"<< _model << endl
                         << "Color:\t"<< _color << endl
                         << "Doors:\t"<< _doors << endl
                         << "Engine:\t"<< _engine_force << " horse\n\n";
                }
};
//############################################//
class vehicle: public movement, public shape {

};
//############################################//

void main(){

vehicle car;

car.power_on();        //start
car.move_on(20,'f');
car.position_();
car.move_on(83,'f');
car.position_();
car.move_on(100,'f');
car.position_();
car.move_on(4,'b');
car.position_();
cout << "\n\n";

car.descripe_();

car.recolor("red");
car.upgread_engine(100);
car.descripe_();
}
```

```
START
MOVE:    -> Moving 20 meter forward
            You moved to 0,20 km

MOVE:    -> Moving 83 meter forward
            You moved to 1,3 km

MOVE:    -> Moving 100 meter forward
            You moved to 2,3 km

MOVE:    -> Moving 4 meter backward
            You back to 1,99 km


Model:   BMW
Color:   white
Doors:   4
Engine:  200 horse

Model:   BMW
Color:   red
Doors:   4
Engine:  300 horse

Press any key to continue . . .
```

الفصل

14

المؤشرات

Pointers

تعريفها:

- عبارة عن متغيرات في الذاكرة تشير إلى عناوين والتي بدورها تشير إلى قيم.
- عبارة عن مصفوفة مفتوحة.

التعامل مع المؤشرات:

- العلامة (&) تستخدم مع المتغيرات والمؤشرات لمعرفة عناوينها.
- العلامة (*) تستخدم مع المؤشرات لمعرفة قيمها.

مثال من الواقع:

لنفرض أنك وأربعة من أصدقاءك ذهبتم إلى المطعم فإنكم ستحجزون طاولة ما بشكل عشوائي، وبفرض أن الطاولة تستوعب أربعة كراسي فقط فإن أي طاولة ستكون مناسبة لجلوس أربعة أشخاص فقط (وكذلك المتغير يفعل "لكنه يحجز ٨ بت على الأقل").
ولنفرض أن زملاءً لك جاءوا إلى المطعم بالصدفة وأردت أن يتناولوا الطعام معك فإن هذا غير ممكن فيجب أن يحجزوا طاولة أخرى عشوائيا ويطلبوا ما يريدونه ويدفعوا فاتورتهم وربما سيكونون في طاولة بعيدة..

ولحل هذه المشكلة اتفقت مع مسئول المطعم بحجز طاولتين متجاورتين ووضعهما بجوار بعض بحيث تستطيعوا تناول الطعام معاً وتكون الفاتورة واحدة، الآن إذا جاء أصدقاء آخرين فإنك لن تستطيع ضم طاولة ثالثة لأنك لم تطلب سوى طاولتين فقط، وإذا طلبت من البداية ضم ثلاث طاولات فربما لا يأتي أصدقائك (وكذلك المصفوفة تفعل)..

لنفترض أنك أردت أن تتخلص من المشكلة السابقة فإنك ستتفق مع أصدقاءك بحجز طاولة وستطلب من مسئول المطعم بأن بضم طاولة إضافية كلما جاء أصدقاء آخرون (وكذلك المؤشرات تفعل).

أرأيت كم هذا مناسب، الآن ستتفادى مشكلة ضم طاولة إضافية ثم لا يأتي زملاءك، وتتجنب الإحراج عندما يأتون ولا تجد لهم مكان..
لنترك المطعم الآن ولنتوجه إلى C++ مرة أخرى لنعقد مقارنة دقيقة..

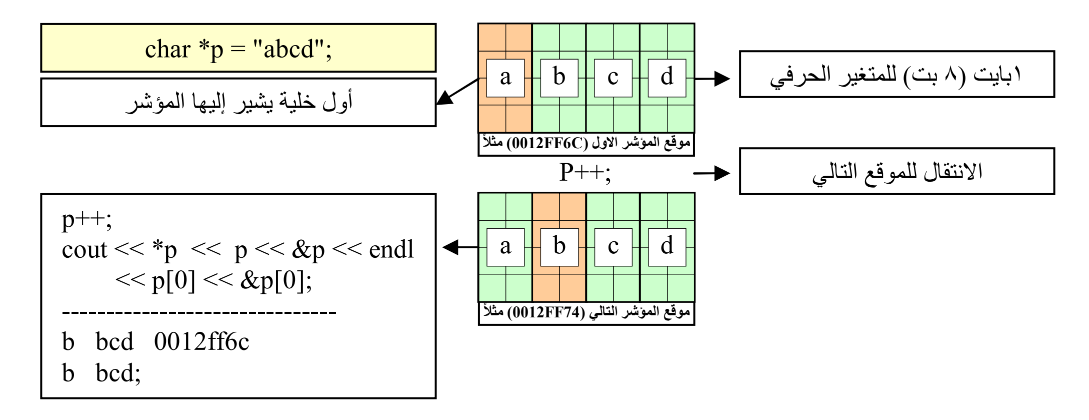
مقارنة بين المتغيرات والمصفوفات والمؤشرات:

وجه المقارنة	الطريقة	ملاحظات
المتغير	int x;	يتم حجز موقع عشوائي "عنوان" في الذاكرة
	x = 9;	توضع القيمة 9 في عنوان المتغير في الذاكرة
	cout << x;	سيتم طباعة قيمة المتغير x.
	cout << &x;	سيتم طباعة عنوان المتغير x في الذاكرة (طباعة رقم سداسي عشر يدل على موقع تقاطع الصف والعمود في الذاكرة RAM).
المصفوفة	int a[3];	يتم حجز ثلاثة مواقع متجاورة في الذاكرة بشكل عشوائي
	a[0] = 7; a[1] = 4; a[2] = 1;	يتم وضع كل قيمة فيما يقابلها من المواقع المحجوزة.
المؤشر	int *p;	يتم حجز موقع عشوائي في الذاكرة "مثل المتغير"
	p = x;	يسبب خطأ يوقف البرنامج لأن المؤشر يحتاج عنوان وليس قيمة متغير.
	int *p = x;	يسبب خطأ يوقف البرنامج لأن المؤشر يحتاج عنوان وليس قيمة متغير.
	p = &x;	يأخذ المؤشر p عنوان المتغير x وبالتالي تصبح قيمة p هي قيمة x، لان المؤشر p أصبح يؤشر إلى عنوان x.

	cout << *p;	طباعة القيمة التي يشير إليها المؤشر p، ويسبب خطأ إذا لم تكن هناك قيمة في الموقع (يجب إسناد قيمة للمؤشر قبل طباعته[1]).
	cout << p;	سيتم طباعة موقع المؤشر p "عنوانه في الذاكرة.
	int *p = &x; *p = 100; cout << x;	يغير المؤشر قيمة المتغير x وبالتالي سيتم طباعة "100" وليس "9"
	cout << &p;	إذا لم يتم إسناد متغير للمؤشر فإنه سيطبع عنوان المؤشر. إذا تم إسناد قيمة المتغير للمؤشر "p = &x;" فإنه سيطبع عنوان المتغير x. (عنوان أول خلية في المؤشر "عنوان ابتدائي"). إذا كان المتغير p حرفي فإنه سيطبع عنوان المؤشر سواءً لم تسند له قيم أو تم إسناده قيم (char *p = "abc";) ولكن إذا تم إسناده متغير (char *p = x;) فإنه سيطبع عنوان المتغير x.
	int *p = &x; p++; cout << p; p--; cout <<p;	(p++) سيتم طباعة رقم عشوائي لأن المؤشر انتقل خطوة للأمام للخلية التالية ولم يضع أي قيمة فيها ويتغير عنوان المؤشر بزيادة 8 بايت بالنظام العشري [0012FF7C + 8 = 0012ff80] لأن نوع البيانات integer يحجز 8 بايت. (p--) الرجوع إلى الخلية السابقة التي فيها قيمة.. وبالتالي سيتم طباعة قيمة x مجدداً.
	char *p = "abc";	يمكن للمتغير النصي أن يسند إليه قيم مباشرة.
	char *p; p = "abc";	يتم إسناد القيم إلى المؤشر مباشرة.
	char *t;	إنشاء مؤشر من النوع الحرفي
	t = "abcd";	الأنواع الحرفية تقبل إسناد قيمة لها بشكل مباشر "دون الاستعانة بمتغير آخر كما هو الحال في النوع العددي". في هذه الحالة فإن المؤشر يعمل كمسئول المطعم فإنه يحجز لكل حرف مساحة مجاورة في الذاكرة بعدد الحروف، تذكر أن كل حرف يحجز 1 بايت (٤ بت) وبالتالي فإن أربعة حروف ستحجز 4 بايت (١٦ بت) متجاورة.
	cout << t;	سيطبع "abcd"
	t++; cout << t;	سيطبع "bcd"
	cout << *t;	سيطبع "a" فقط لأنه سيطبع قيمة أول خلية يشير إليها المؤشر وإذا أردنا الانتقال للقيمة في الخلية التالية نزيد عدد t بمقدار واحد (t++)، كيف يعرف الحاسوب ذلك (لأن المؤشر يشير إلى الخلية التي بعده).
	t++; cout << *t;	سيطبع "b"
	t--; cout << *t;	سيطبع "a" لأننا رجعنا خطوة للخلف، والرجوع خطوة أخرى أو التقدم خطوات زيادة للأمام قد تصادف مواقع خالية في الذاكرة "Null" وبالتالي لن يطبع شيء (في المؤشرات الرقمية سيطبع رقم عشوائي).
	cout << &t[0];	سيطبع القيم ابتداءً من الخلية الأولى (abcd).
	cout << &t[1];	سيطبع القيم ابتداء من الخلية التالية إلى نهاية سلسلة الحروف (bcd).
	cout << t[0];	سيطبع القيمة التي في الخلية الأولى (a).
	cout << t[1];	سيطبع القيمة في الخلية الثانية (b).
	t++; cout << t[0];	سينتقل المؤشر إلى الخلية التالية ويعتبرها الأولى لذا سيطبع القيمة الثانية (b).
	t++; cout << &t[0];	سينتقل المؤشر إلى الخلية التالية ويعتبرها الأولى لذا سيطبع القيم ابتداء من الخلية الثانية إلى نهاية سلسلة الحروف (bcd).
	t++; p = "ahmed";	سيتقدم المؤشر خلية واحدة ثم سيضيف العبارة "ahmed" ابتداءً من الخلية التي يقف عليها وبعدد الحروف (خمسة خلايا) [2].

(1) عند إنشاء مؤشر فإنه لا يشير إلى أي مكان وتكون قيمته "Null" إلا بعد إعطاءه عنوان متغير موجود بالفعل أو إسناده قيمة حرفية في حالة المؤشر الحرفي "char".

(2) سيتم حذف الحروف الموجودة ابتداءً من الخلية التي يقف عليها المؤشر.



مثال شامل للتعامل مع المؤشرات:

```
int i = 1;
int *p = &i;
//p++;
cout << &p <<" "<< *p <<" "<< p <<" "<< p[0] << " " << p[1]
      << " - " << &p[0] << " " << &p[1] << "\n\n\n";

char *c = "abcd";
for(int i=0; i<=3; i++){
      cout << c[i] << endl;
}

cout << &c <<" "<< *c <<" "<< c <<" "<< c[0] << " " << c[1]
      << " - " << &c[0] << " " << &c[1] << "\n\n\n";

c++;
cout << &c <<" "<< *c <<" "<< c <<" "<< c[0] << " " << c[1]
      << " - " << &c[0] << " " << &c[1] << "\n\n\n";

c++;
cout << &c <<" "<< *c <<" "<< c <<" "<< c[0] << " " << c[1]
      << " - " << &c[0] << " " << &c[1] << "\n\n\n";

cout << &c <<" " << *c<<" "<< c <<" "<< c[0] << " " << &c[0] << endl;
```

مميزات وعيوب هياكل البيانات:

النوع	الطريقة	المميزات	العيوب
المتغير	يحجز موقع للقيمة	سهل التعريف والاستدعاء والإسناد.	لكل قيمة يجب تعريف متغير جديد يصعب الحصول على المتغيرات فكل متغير يحتاج إلي استدعاءه باسمه.
المصفوفة	تحجز عدة مواقع متجاورة متساوية الطول والنوع	توفر طريقة سهلة لإدخال القيم واستخراجها بالدوران مثل for	يجب تحديد حجم المصفوفة مسبقاً ولا يمكن تحديد حجمها أثناء التشغيل. كل متغيرات المصفوفة الواحدة يجب أن تكون من نفس النوع لا يمكن زيادة أو تقليص حجمها بحسب البيانات الفعلية.
التركيب	يحجز عدة مواقع متجاورة غير متساوية الطول أو النوع حسب الرغبة	نفس مميزات المصفوفة يمكن عمل تركيب داخل تركيب توفر طريقة سهلة لتجميع متغيرات مختلفة في تركيب واحد يسهل التعامل معه يمكن عمل مصفوفة من التركيب	تبقى مشكلة مصفوفة التركيب حيث لا يمكن زيادة أو تقليص حجمها لا يمكن تحديد حجم المصفوفة أثناء التشغيل.

المؤشر	يحجـز عنـوان ويـضع فيـه قيمـة، ويـستطيع التحـرك لوضـع قيمـة فـي الموقع المجاور أو استدعاء قيمة منه.	يستطيع التحرك في الذاكرة وتعبئتها بالقيم حسب الطلب.	صعوبة المؤشرات فهي تتعامل مع العناوين وليس القيم
		لا يحجز إلا المساحة المطلوبة ويمكن زيادة حجمه وتقليصه	خطورة التعامل مع المؤشرات
		يضيف إمكانات هائلة للمصفوفات و التراكيب وحتى الكائنات	صعوبة اكتشاف أخطاء المؤشرات
		يمكـن تعريـف مؤشـر مـن نـوع مـصفوفة وبالتالي يمكن تعيين حجمها أثناء التشغيل.	تستطيع تحديد حجم المصفوفة من نـوع مؤشـر أثنـاء التـشغيل لكن بعد تحديده لا يمكن زيادته أو تقليصه.

لا تدخل الكائنات والدوال ضمن المقارنة لأن الكائنات تحتوي نفس عيوب التركيب، ولأن الدوال لا تضيف إمكانيات جديدة للمتغيرات "غير أنه يمكن تعريف متغير بنفس الاسم داخل عدة دوال في نفس الوقت (متغير محلي)".

إمكانيات المؤشرات[1]:

ـ تعريف مصفوفة متغيرة الحجم:

```
int count = 0;
cout << "Enter array count: ";
cin >> count ;

char *c = new char[count];

for(int i=0; i<count; i++){
    cin >> c[i];
}
for(int i=0; i<count; i++){
    cout << c[i] << ends;
}
cout << endl;
```

ـ تعريف مصفوفة متغيرة الأبعاد:

```
int rows,cols;
cout << "Enter rows count: ";  cin >> rows ;
cout << "Enter cols count: ";  cin >> cols ;

int **Array = new int *[rows] ;

for (int p=0 ; p < rows ; p++)
{    Array[p] = new int[cols];   }

for (int r=0 ; r < rows ; r++)
{   for (int c=0; c < cols; c++)
    {   cout << "Enter array[" << r << "][" << c << "] value: ";
        cin >> Array[r][c];
    }   cout << endl;
}

for (int r=0 ; r < rows; r++)
{   for (int c=0; c < cols; c++)
    {    cout << "[" << Array[r][c] << "]";    }
         cout << endl;
}
```

(1) الإكسير في c++ للكاتب "سلطان محمد الثبيتي".

حذف المؤشرات(1):

تحدث المشاكل في البرامج الكبيرة عندما تنسى إسناد قيمة للمؤشر فيستمر المؤشر بحجز مساحة فارغة وعندما يحاول الحاسوب حجز هذه المساحة لأنها فارغة لمتغير آخر فقد يحدث خلل يوقف البرنامج أو حتى الحاسوب عن العمل.. وبالتالي يجب إسناد قيم للمؤشرات وعند الانتهاء من استخدام المؤشر يجب إلغاؤه.

طريقة حذف المؤشر:

```
int *p = &x;
```

يتم إسناد القيمة "صفر" للمؤشر:

```
p = 0;
```

فائدة:

طالما أنه يمكننا معرفة عنوان المتغير ونستطيع تغيير القيمة الموجودة في ذلك العنوان فإن ذلك سيعطينا قدرات جديدة، فمثلاً نستطيع أن نرسل عنوان متغير لدالة أو حتى إجراء والذي بدوره يقوم بتغيير قيمة العنوان بقيمة جديدة، وبالتالي يتم تحديث قيمة المتغير، كلا الطريقتين التاليتين تقوم بنفس العمل.

- باستخدام العناوين:

```
void sum( int &q, int b, int c){
        q = b + c;
}
main(){
        int a = 0;

        sum(a,  2 , 3 );
        cout << a;  // print 5
}
```

- باستخدام المؤشرات:

```
void minus( int *w, int b, int c){
        *w = b - c;
}
main(){
        int a = 0;

        minus(&a,  7 , 5 );
        cout << a;  // print 2
}
```

(1) المرجع السابق.

الفصل

15

الملفات

Files

مقدمة:

إن استخدام الذاكرة الظاهرية (Ram) ومعرفة التعامل معها وكيفية التخزين فيها واسترجاع البيانات منها لا يكفي لعمل برنامج متكامل، حيث أن الذاكرة الظاهرية تحذف البيانات بمجرد إغلاق البرنامج.. لذا فلا بد من وسيلة لحفظ البيانات بشكل دائم. تعتبر الملفات وسيلة ضرورية لحفظ المعلومات بشكل دائم، وتتيح لنا تخزين البيانات في القرص الصلب والرجوع إليها في أي وقت للاطلاع والإضافة والتعديل.

مقارنة:

النوع	HD	RAM
الشكل	Hard Disk	Random Access Memory
الاسم	القرص الصلب	ذاكرة الوصول العشوائي
الوظيفة	يتم استخدام هذه الذاكرة لحفظ البيانات بصورة ملفات بشكل دائم حتى بعد إعادة تشغيل الجهاز.	تقوم البرامج باستخدام هذه الذاكرة لتخزين قيم المتغيرات وإجراء العمليات عليها، وتفقد هذه القيم المخزنة بمجرد إغلاق البرنامج.
الحجم	حجمهم كبيرة يتراوح من 40GB إلى 1TERA.	حجمها صغير تتراوح من 128mb إلى 4GB
التحكم	يمكنك مشاهدة الملفات وفتحها واستعراض البيانات داخلها.	لا يمكنك التحكم بها ولا حتى مشاهدة البيانات فيها
السرعة	بطيئة في خزن واسترجاع البيانات	سريعة جداً في خزن واسترجاع البيانات
الإتاحة	كبيرة الحجم ولا تبطئ عمل الحاسوب، وهي متاحة لخزن البيانات لكنها تمتلئ بالبيانات مع الوقت ولا يمكن تخزين بيانات إضافية فيها. وينبغي شراء قرص جديد أو حذف البيانات منها يدوياً.	لا تمتلئ بالبيانات لأنها تحذف ما بداخلها أولا بأول، فتكون متاحة لكل التطبيقات لاستخدامها لكن لها سقف محدود لا يمكنها تخطيه هو حجمها فالذاكرة الصغيرة تبطئ عمل الحاسوب لأنها تتعامل مع قدر صغير من البيانات فتنتظر البيانات في طوابير حتى تفرغ الذاكرة.

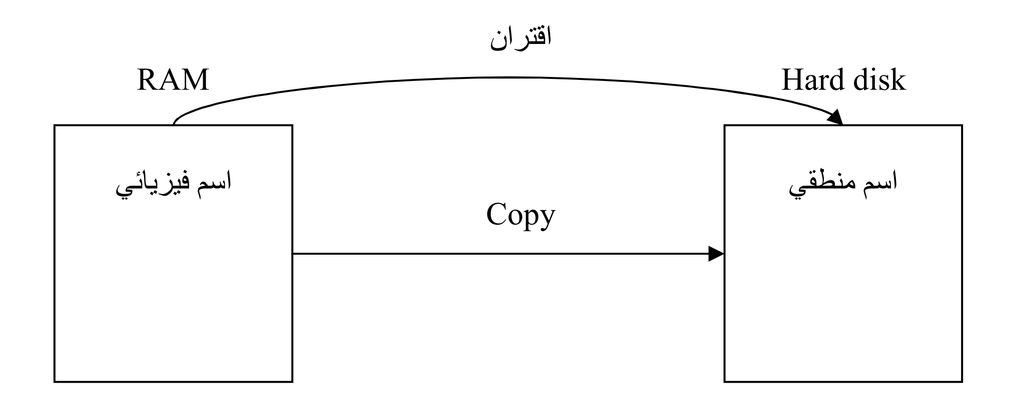
كيف يتم تخزين الملفات:

ـ مثال من الواقع:

عندما تقوم بالكتابة في برنامج word فإن كل التغييرات التي تقوم بها تكون في الذاكرة العشوائية ram وعندما تقوم بالحفظ (ctrl+s) فإن البيانات تنقل من الرام إلى القرص الصلب فكيف يحدث ذلك؟

تمتلك الملفات هيئتين مختلفتين بناء على الوسط الذي تكون فيها ولذا فلها اسمين:

- **الاسم الفيزيائي:** وهو تمثيل في الذاكرة والبرنامج لاسم الملف.
- **الاسم المنطقي:** وهو تمثيل حقيقي في القرص الصلب لاسم الملف يحتوي على الاسم مع المسار والامتداد.



- عملية النسخ:

عندما نقوم بحفظ العمل فإنه لا يتم نسخ البيانات من الرام إلى القرص الصلب بشكلها الحالي ولكن يجب تمثيلها على شكل نطاق يحجز في الذاكرة.
النطاق عبارة عن مؤشر من نوع *char يكون حجمه في البداية 0 ويستوعب كافة البيانات في الذاكرة وبالتالي يتم إسناد قيمة المؤشر "الفيزيائية" إلى القرص الصلب كقيمة "منطقية" وكأننا نقول y = x.

- خطوات الحفظ لملف:

١) تضمين مكتبة fstream.h وهي تتكون من صنفين ifstream و ofstream.

```
include <fstream.h>
```

٢) إنشاء ملف (فتح ملف للطباعة)

```
ofstream اسم فيزيائي (" الاسم المنطقي للملف مع المسار والامتداد ");
```

٣) الطباعة إلى الملف

```
ff << "some text here..";
```

٤) إغلاق الملف.

```
ff.close();
```

ملاحظات:

```
ofstream ff("c:\\ali.txt");
```

- في المثال نلاحظ أنه يشبه طريقة المشيد الافتراضي (المبدل) في الكائنات فهو يمرر القيمة ويشغلها.
- تسمى هذه طريقة الفتح للكتابة وهي تقوم بإنشاء الملف إن لم يكن موجوداً.

```
ff << "some text here..";
```

- لأن الصنف ofstream من النوع ostream فإنه يستخدم نفس معامل الاخراج >>.
ويقصد بهذا السطر أن يتم تخزين الجملة ..some text here في الملف المحدد في المسار c:\ali.txt.

```
ff.close();
```

- لا يتم إنشاء الملف في القرص الصلب إلا عند كتابة هذا السطر فهو يقوم بنسخ النص من الرام إلى القرص الصلب ويقوم أيضاً بإغلاق الملف.

كيف تتم قراءة الملفات:

لإجراء أي تعديل على ملف word الذي أكتبه يجب أن يتم تحميله إلى الذاكرة الرام (قراءة الملف) ثم إجراء التعديلات عليه ومن ثم إعادة حفظ الملف.

ـ خطوات القراءة من الملف:

١) فتح الملف للقراءة.

```
ifstream hh("الاسم المنطقي مع المسار والامتداد");
```

٢) القراءة من الملف:

ـ قراءة سطر واحد وإسناد قيمته لمتغير.

```
char p;
hh>>p;
```

ـ قراءة جميع السطور.

تحتاج لقراءة جميع السطور إلى دالة دوران حيث ستقرأ سطر واحد فقط في كل لفة.

```
While(!hh.eof()){
        hh>>p;
        cout << p << endl;
}
```

٣) إغلاق الملف

```
hh.close();
```

ـ ملاحظات:

```
ifstream hh("c:\\ali.txt");
```

يجب أن يتم تحديد ملف موجود في القرص الصلب للقراءة منه.

```
char p;
hh>>p;
```

يتم إسناد محتويات السطر في المتغير p .

```
While(!hh.eof()){
        hh>>p;
        cout << p << endl;
}
```

تعني هذه العبارة أنه طالما لم يصل المستند الى نهايته (!hh.eof()) فقم بإسناد قيمة السطر الحالي للمتغير p واطبعها على الشاشة.

مثال لتخزين البيانات في ملف:

	الكود		الشرح
1	main(){	١	
2	Char i,x,c,t	٢	تعريف ٤ متغيرات
3	cout << "enter your number: "; cin >> i;	٣	طلب قيمة (الرقم) من المستخدم
4	cout << "enter your name: "; cin >> x;	٤	تخزين القيمة في المتغير
5	cout << "enter your country: "; cin >> c;	٥	طلب قيمة (العمر) من المستخدم
6	cout << "enter your city: "; cin >> t;	٦	تخزين القيمة في المتغير
7		٧	
8	ofstream ff("c:\\ali.txt");	٨	فتح ملف للكتابة
9	ff << i << x<< endl	٩	كتابة نصوص وقيم المتغيرات إلى ملف
10	<< c << t << endl;	١٠	
11	hh.close();	١١	إغلاق الملف
12	}	١٢	

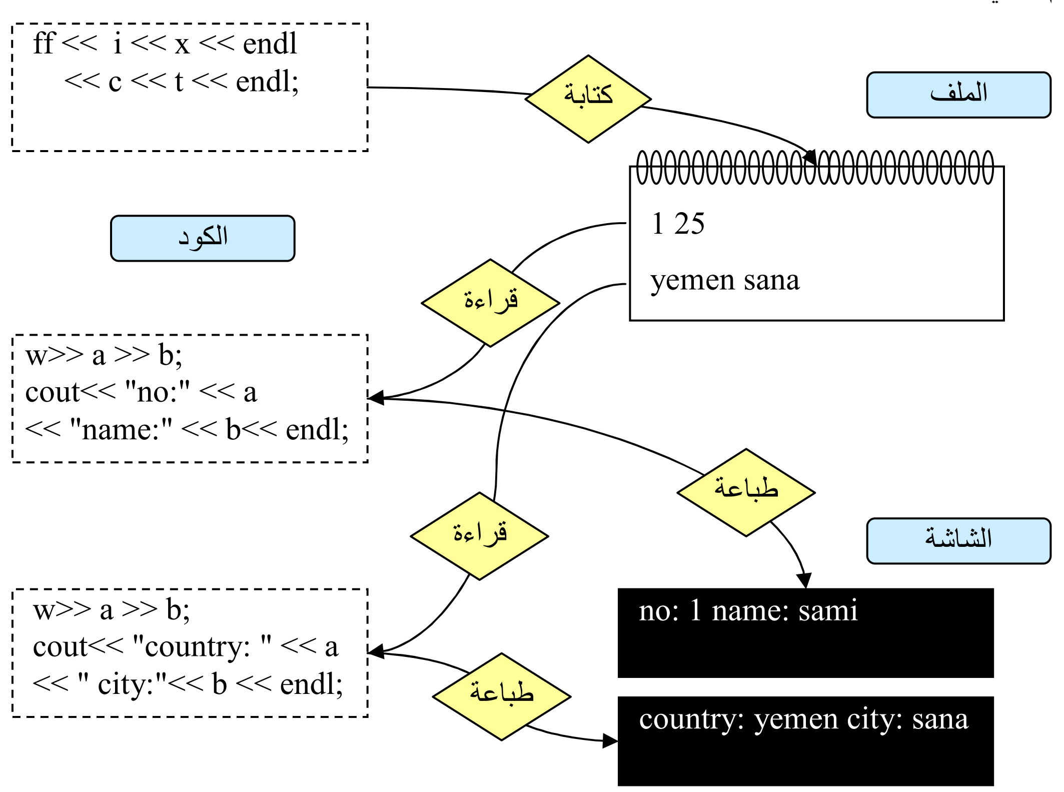
مثال لقراءة البيانات من ملف:

الكود		الشرح
1.	`main(){`	١
2.	`char a,n;`	٢ تعريف متغيرين
3.	`ifstream w("c:\\ali.txt");`	٣ فتح ملف للقراءة
4.	`w>> a >> b;`	٤ تخزين القيمة الأولى في الملف إلى المتغير a والثانية للمتغير b
5.	`cout << "no:" << a << "name:" << b << endl;`	٥ طباعة القيمة على الشاشة بشكل منسق
6.	`w>> a >> b;`	٦ تخزين قيم السطر الثاني
7.	`cout << "country:" << a << "city:" << b << endl;`	٧ طباعة قيم السطر الثاني
8.	`hh.close();`	٨ إغلاق الملف
9.	`}`	٩

ملاحظات:

عند الطباعة إلى ملف يتم فصل كل قيمة عن الأخرى بمسافة وذلك لأنه عند القراءة من الملف فإن دوال القراءة تستطيع تمييز كل كلمة مستقلة وبالتالي يمكن إسناد كل كلمة مفصولة بمسافة إلى متغير آخر.

انظر إلى السطر رقم 5 تم إسناد الرقم للمتغير a والاسم للمتغير b لأننا فصلنا بينهما بمسافة في الملف، انظر الى الرسم التالي:



لاحظ أنه كلما تم استخدام الجملة (w>> a >> b;) فسيتم تخزين قيم السطر التالي وبالتالي سيتم طباعة بيانات السطر التالي..

مثال شامل:

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class files{
private:
	char path2file[255];

public:
	files(char p[]){
		strcpy(path2file, p); // نسخ اسم الملف إلى المتغير
	};
	void write(string s ){
		ofstream o(path2file, ios::out); // استبدال محتويات الملف
		o << s;
		o.close();
	};
	void append(string s ){
		ofstream o(path2file, ios::app); // كتابة في نهاية الملف
		o << s;
		o.close();
	};

	string read(int line){
		ifstream r(path2file, ios::in); // قراءة من الملف

		char x[255];
		int u=0;
		while( r.getline(x ,255)!=0 ){  // قراءة سطر كامل
			if(u >= line){    break; }
			u++;
		}
		if(line>u){cout << ".........\n";}
		r.close();
		return x;
	};

	int count(){
		ifstream r(path2file, ios::in);
		int i=0;
		char x[255];
		while( r.getline(x ,255)!=0 ){
			i++;
		}
		return i;
	}
};

void main(){
	files f("new.txt");                    // إنشاء ملف جوار البرنامج
	f.write("hello word\nwelcome\n");      // كتابة واستبدال
	f.append("yahoo\n");                   // إلحاق في نهاية الملف
	cout << f.read(2) << endl;             // طباعة السطر الثالث
	cout << f.read(3) << endl;
	cout << f.read(0) << endl;             // طباعة السطر الأول
	cout << f.read(4) << endl;
	cout << f.count() << endl;             // طباعة عدد السطور
}
```

الفصل
16

ملحقات
Supplements

أنواع البيانات:

المدى الأدنى للقيم إلى	 من	الحجم بالبايت	الحجم بالبت	النوع
127	127	1	8	**char**
255	0	1	8	unsigned **char**
127	127	1	8	signed **char**
32,767	32,767	2 or 4	16 or 32	**int**
65,535	0	2 or 4	16 or 32	unsigned **int**
نفس مدى int		2 or 4	16 or 32	signed **int**
32,767	32,767	2	16	short **int**
65,535	0	2	16	unsigned short **int**
نفس مدى short int		2	16	signed short **int**
2,147,483,647	2,147,483,647	4	32	long **int**
نفس مدى long int		4	32	signed long **int**
4,294,967,295	0	4	32	unsigned long **int**
ست خانات عشرية		4	32	**float**
عشر خانات عشرية		8	64	**double**
عشر خانات عشرية		10	80	long **double**

المحارف الخاصة:

Code	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\0	Null
\\	Backslash
\v	Vertical tab
\a	Alert
\?	Question mark
\N	Octal constant (where N is an octal constant)
\xN	Hexadecimal constant (where N is a hexadecimal constant)

العمليات المنطقية:

p	q	p && q	p || q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

أولوية العمليات:

Highest	( ) [ ] | > .
	! ~ ++ – – (type) * & sizeof
	* / %
	+ |
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= += |= *= /= etc.
Lowest	,

المراجع :

محاضرات الاستاذ / بسام الهاملي

The Complete Reference C++ Third Edition

تطبيق UML التحليل والتصميم بالمنحى للكائن باستخدام UML ترجمة وإعداد "خالد الشقراوي".

الإكسير في c++ للكاتب سلطان محمد الثبيتي.

الفهرس
Index